National Research Council Canada
Conseil national de recherches Canada
Division of Electrical Engineering
Division de génie électrique
ERB-1000

M*: A Methodology for Information System Analysis and Database Design in Production Environments

A. Di Leva, F. Vernadat, and D. Bidar

December 1986

87-04603/

Copyright 1986 by
National Research Council of Canada

Permission is granted to quote short excerpts and to reproduce figures and tables from this report, provided that the source of such material is fully acknowledged.

Copyright 1986 par
Conseil national de recherches du Canada

Il est permis de citer de courts extraits et de reproduire des figures et tableaux du présent rapport, à condition d'en indiquer clairement la source.

Additional copies are available free of charge from:

Editorial Office, Room 351
Division of Electrical Engineering
National Research Council of Canada
Ottawa, Ontario, Canada
K1A 0R6

Des exemplaires supplémentaires peuvent être obtenus à l'adresse ci-dessus.

M*: A METHODOLOGY FOR INFORMATION SYSTEM ANALYSIS AND DATABASE DESIGN IN PRODUCTION ENVIRONMENTS

A. Di Leva, F. Vernadat, and D. Bizier

ERB-1000
NRCC No. 26204

DECEMBER 1986

Table of Contents

ABSTRACT v
RESUME vi
1. INTRODUCTION 1
2. BASIC TERMINOLOGY AND FUNDAMENTAL CONCEPTS 3
2.1 General Terminology 3
2.2 Data Versus Information 3
2.3 Database Technology 4
2.3.1 The Logical Level of Database Systems 4
2.3.2 The Physical Level of Database Systems 5
3. PRODUCTION SYSTEM ANALYSIS AND DATABASE DESIGN 6
3.1 Production System Analysis 6
3.2 Database Design 6
3.3 Methodological Framework for Implementing Production and CIM Databases 7
4. THE M* GENERAL ARCHITECTURE 9
5. THE INTEGRATED MODEL (OR M* TOOL-BOX) 11
5.1 Introduction 11
5.2 The Organizational Level 11
5.2.1 The Functional Model 11
5.2.2 The Information Model 13
5.2.3 The Dynamics Model 14
5.3 The Conceptual Level 14
5.3.1 The Data Model 14
5.3.2 The Operation Model 17
5.3.3 The Process Model 19
5.4 The Implementation Level 20
5.4.1 The Data Definition Language (DDL) 21
5.4.2 The Data Manipulation Language 21
5.4.3 The Process Definition Language 22
5.5 Conclusion 23
6. THE CASE STUDY 25
7. Phase 1: ENTERPRISE MODELLING AND ANALYSIS 27
7.1 Introduction 27
7.2 Task 1.1: AS-IS Analysis 27

7.2.1 Task 1.1: Input Description 27
7.2.2 Task 1.1: Output Description 28
7.2.3 Task 1.1: Description of Activities 28
7.2.4 Top-down Analysis 29
7.2.5 Bottom-up Analysis and Diagnosis 30
7.3 Task 1.2: TO-BE Analysis 30
7.3.1 Task 1.2: Input Description 33
7.3.2 Task 1.2: Output Description 33
7.3.3 Task 1.2: Description of Activities 33
7.4 Task 1.3: Environment Specification 34
7.4.1 Task 1.3: Input Description 35
7.4.2 Task 1.3: Output Description 35
7.4.3 Task 1.3: Description of Activities 36
7.4.4 DB Committee Set Up 36
7.4.5 Requirements Collection Planning 39
7.4.6 Requirements Collection 40
7.5 Conclusion 40
8. Phase 2: CONCEPTUAL DESIGN 42
8.1 Introduction 42
8.2 Task 2.1: Form Analysis 43
8.2.1 Task 2.1: Input Description. The Form Model 43
8.2.2 Task 2.1: Output Description 44
8.2.3 Task 2.1: Description of Activities 44
8.3 Task 2.2: Requirements Analysis 48
8.3.1 Task 2.2: Input Description 49
8.3.2 Task 2.2: Output Description 49
8.3.3 Task 2.2: Description of Activities 49
8.4 Task 2.3: View Design 50
8.4.1 Task 2.3: Input Description 52
8.4.2 Task 2.3: Output Description 52
8.4.3 Task 2.3: Description of Activities 52
8.5 Task 2.4: View Integration 56
8.5.1 Task 2.4: Input Description 56
8.5.2 Task 2.4: Output Description 56
8.5.3 Task 2.4: Description of Activities 56
8.6 Validation of Data Schemata in M* 58
8.7 Conclusion 59
9. Phase 3: IMPLEMENTATION DESIGN 60
9.1 Introduction 60
9.2 Task 3.1: Logical Design 60

9.2.1 Task 3.1: Input Description 61
9.2.2 Task 3.1: Output Description 62
9.2.3 Task 3.1: Description of Activities 63
9.3 Task 3.2: Physical Design 67
9.3.1 Task 3.2: Input Description 70
9.3.2 Task 3.2: Output Description 70
9.3.3 Task 3.2: Description of Activities 70
9.4 Conclusion 74
10. CONCLUSION 75
11. ACKNOWLEDGEMENTS 75
12. REFERENCES 76
13. APPENDIX: LIST OF M* FORMS 79

ABSTRACT

In this report, a methodology, called M*, for information system analysis and database design of production environments is presented in detail. The originality of M* is to recognize the importance of integrating factory analysis methods for data and user requirements definition with logical database design methods for database specification. The M* methodology is based on the Italian DATAID methodology for administrative database design. M* is a pragmatic, complete, manual methodology involving three major phases: Enterprise Modelling and Analysis which is the phase devoted to accurate analysis of the structure of the enterprise and precise definition of the scope of the future database application; Conceptual Design which is the phase concerned with data modelling of both static and dynamic properties of enterprise information, conceptual schema definition by means of the Extended Entity-Relationship (EER) model using a bottom-up approach, and enterprise data processing modelling using an extension of Petri nets; and lastly Implementation Design which is the phase dealing with adaptation of the conceptual schema to the data model specifications of the physical database system used for application development and tuning performances of the physical system. In this document, the implementation design phase is discussed in the case of DBS/R, a relational database system developed at the National Research Council of Canada. The three methodological phases are explained and illustrated by means of an example related to Flexible Manufacturing Systems (FMS) for mechanical parts.

RESUME

Dans ce rapport, on présente en détail une méthodologie, appellée M*, pour l'analyse des systèmes d'information et la conception des bases de données dans les environnements de production. L'originalité de M* est de reconnaître l'importance de l'intégration des méthodes d'analyse d'entreprise pour la définition des besoins en données avec les méthodes de conception logique de bases de données pour la spécification de base de données. La méthodologie M* est basée sur la méthodologie italienne DATAID qui sert à la conception de bases de données administratives. M* est une méthodologie manuelle, pragmatique et complète impliquant trois phases principales: la Modélisation et l'Analyse d'Entreprise qui sert à faire une analyse précise de la structure de l'entreprise et à définir clairement l'étendue de la base de données à mettre en application; l'Analyse Conceptuelle qui est la phase de modélisation des données en tenant à la fois compte des propriétés statiques et dynamiques des éléments d'information de l'entreprise, de définition du schéma conceptuel au moyen du modèle entité-association augmenté (EER model) en adoptant une approche ascendante, et la modélisation du traitement des données de l'entreprise en utilisant une extension des réseaux de Pétri; et finalement l'Analyse de l'Implantation qui est la phase concernant l'adaptation du schéma conceptuel aux spécifications du modèle de données du système de base de données physique utilisé pour le développement de l'application, et l'ajustement des performances du système physique. Dans ce document, on présente la phase d'analyse de l'implantation pour DBS/R, un système de bases de données relationnelles développé au Conseil National de la Recherche du Canada. Les trois phases de la méthodologie sont expliquées et illustrées au moyen d'un exemple relatif aux systèmes d'ateliers flexibles (Flexible Manufacturing Systems FMS) pour la fabrication de pièces mécaniques.

M*: A Methodology for Information System Analysis and Database Design in Production Environments

Antonio DI LEVA (*), François VERNADAT, and Daniel BIZIER (**)

1. INTRODUCTION

To become more productive and to remain competitive, more and more companies must optimize and automate their information system and their operations in implementing concepts of Computer-Integrated Manufacturing (CIM). The concept of CIM (1, 2), i.e., the use of computer assistance at any levels of the company, from order entry and product design to product manufacturing and shipping, cannot be implemented and achieved if it does not rely on a well-structured database tailored and tuned for the specific operations of the company (3). In other words, database technology (4-6) and logical database design in particular (7, 8) are central to the CIM concept.

Furthermore, CIM is not a technology that can be purchased; it is a company's strategy that must be achieved. This means that before starting any automation project in the company within the scope of augmenting overall productivity, increasing product quality, reducing product leadtimes, reducing operation costs, recovering financial health, or whatever, a good understanding of the current status of the company must be pursued, a strategic plan firmly established, automation opportunities clearly identified, their impact on the whole production system well-understood, and the required supporting information system precisely defined. Then, components of the CIM system can be put in place with respect to one major objective that must be continuously kept in mind: integration (9). Once again database technology can play an important role in this integration.

It is now accepted that any well-designed CIM system must be developed using a top-down approach (involving strategic plan formulation, requirements analysis and specifications, system analysis and design, and conceptual database design) and implemented using a bottom-up approach (involving database implementation, equipment installation, application development, system integration, test and maintenance, and operational phase) (10).

In previous papers dealing only with the top-down phase of CIM system development, some modelling and analysis tools for factory analysis have been presented and discussed (11-13), the role of database technology for production system automation and integration has been emphasized and is widely recognized as being one of the fundamental components of any CIM system (14), and an appropriate methodological framework for the implementation of CIM databases has been recently proposed (15).

Several structured system design and analysis methodologies have been developed around the world and applied to production systems, some of them in the framework of large projects financed by governments or industry and others in universities. Three of these have greatly influenced our work: the SADT method (16) and the data-flow diagram technique (17), the IDEF method (11) developed as part of the US IPAD project,

(*) Permanent address: Universita' di Torino, Dipartimento di Informatica, via V. Caluso 37, 10125 Torino, Italy

(**) Daniel BIZIER, Ing., MBA

NRCC 26204

and the GRAI method (12) elaborated at the University of Bordeaux. All these methodologies are mostly system analysis tools and do not take into account the database aspect of the enterprise.

In this report, we stress the importance of integrating factory analysis methodologies with database design methodologies for a better design and integration of automated production systems or CIM systems. The basic steps of a fully integrated methodology, called M* and under development at the National Research Council of Canada, will be described and illustrated. The aim of the M* methodology is to support the analysis of the information system of a production organization and the design of the database environment which will be used in the day-to-day activities of the organization. Although most of this work has been developed in the framework of manufacturing and production systems, the methodology remains applicable to any application environments.

With the M* methodology, the complete design of a database application is decomposed into three main design phases, according to the general methodological framework for database design: (a) *Enterprise Modelling and Analysis*, (b) *Conceptual Design*, and (c) *Implementation Design*.

The methodology M* is a manual methodology dealing with centralized databases. It is an experimental product that must be improved on the basis of practical experience. In this document, we will illustrate its application by means of a real-world case study concerning a Flexible Manufacturing System (FMS) environment. The example will be used to illustrate the presentation of the approach and can be used to allow a comparison with other approaches described in the literature (18).

The structure of this report is the following. First, Section 2 will introduce some basic concepts and terminology frequently used throughout the report. In Section 3, the importance of integrating production system analysis methods with database design methods is emphasized. In Section 4, the general architecture of the M* methodology is explained. In Section 5, the integrated model supporting M*, that is the set of methods, models, and techniques used in the three methodological phases, is presented. In Section 6, a case study dealing with FMS (Flexible Manufacturing System) and used for illustrative purposes throughout the report is described. Sections 7 to 9 give the detailed description of the three methodological phases. Each phase will be broken down into one or more tasks. Each task may involve one or more steps. For each task, the description of its inputs, the description of its outputs, and the description of its activities will always be considered in this order. Finally, Section 10 states the conclusions. Furthermore, the reader will find an Appendix at the end of the report containing examples of the forms used in the methodology.

2. BASIC TERMINOLOGY AND FUNDAMENTAL CONCEPTS

This report deals essentially with functional analysis and modelling of production systems and with database design methods. Therefore, some concepts used in this specific context must be defined.

2.1. General Terminology

An *object enterprise* or *object organization* is a real-world system (company, production process, set of activities, ...) subject to be the focus of an analysis.

Object enterprises or organizations are usually composed of one or more functions. A *function* is a set of activities or operations executed to perform some task relevant to some enterprise. Functions can be broken down into sub-functions if necessary. Functions and sub-functions are usually defined in terms of their inputs, their outputs, their constraints, and their operating procedures.

Functions are related to one another or communicate by means of real-world objects. *Real-world objects* are usually composed of general objects, or simply objects, and information objects.

Objects of an enterprise may represent anything (materials, things, documents, data, etc.) which can be exchanged, input, or output by functions.

Information objects are real-world objects relevant to information processing in an enterprise (such as data, documents, messages, data files, forms, verbal orders, etc.).

An *operation* is a basic action performed during the execution of a function.

An *activity* is a set of operations. Activities are performed by users or application programs.

A *user* is a person involved in the execution of activities of a function.

An *application program* is an automatic procedure for the execution of activities of a function.

An enterprise *environment* is a subset of the enterprise composed of users and functions sharing a common part of the information system of the enterprise.

The *information system* of some enterprise is the set of all data and information objects as well as their processing procedures used to support the operations of the enterprise. Information systems are usually not formally structured systems in enterprises. The formalized part is very often the enterprise database system with its related procedures. Thus, a database system must not be confused with an information system. The database is only a part of the information system.

2.2. Data Versus Information

Data and information are two words that are very often used interchangeably in the common language. However, they have different meanings in information processing.

So far, most existing databases deal only with *data* and especially facts. For example, if we consider the information

machine MT1 is idle

The expression "idle" is a fact or a value, not a datum. To have a datum, we need two things: a *fact* and a *meaning*. A fact alone has no meaning. Thus, in the example, one datum is composed of the value "idle" plus the meaning that idle is a possible value for a machine status.

Information is the meaningful interpretation and correlation of some aggregation of data in order to allow one to make decisions. In the example, the information is composed of two data elements: one indicating that we refer to "machine MT1" and the second one indicating that the machine status is "idle".

The distinction and relationship between data and information is illustrated in Fig. 1.

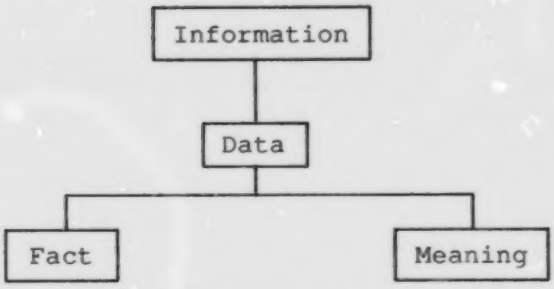


Figure 1: Data Versus Information

2.3. Database Technology

In this Section, we review important concepts of database theory for the sake of completeness of this paper. This discussion is by no means exhaustive. For a full discussion on database technology, we refer the reader to refs. 10, 18, and 19.

A *database system (DBS)* is a computer environment for some enterprise composed of:

- computer hardware (cpu, storage devices, input/output devices, ...)
- some databases, i.e., data on mass storage
- a database management system, i.e., computer software
- application programs executing some enterprise activities
- and users, i.e., the human side of the system

2.3.1. The Logical Level of Database Systems

A *database (DB)* is a large collection of inter-related data in mass storage to be accessed and shared by various users/applications of some enterprise. Although data are usually stored in files on disk, a database must not be confused with a data file. Data in a database can be accessed concurrently by several users or applications, redundancy of data is limited as much as possible, integrity of data can be maintained, security of data can be controlled, data can be queried, and independence of data against program modification can be achieved.

Data in the database are structured according to a *data schema* which is a canonical description of the data object types for a given application with their attributes, relationships, and constraints. The data schema or conceptual schema is a static model (or *intension*) of the logical structure of the database. The database is an instance (or *extension*) of the schema at a given time. The extension of a database is a temporal model of the real-world that the database is supposed to represent. A *subschema or view* is a subset of the conceptual schema adapted for a group of users of the database.

Data schemata are defined in terms of some data models. A *data model* is a set of notational and operational concepts for the expression of database schemata. Classical data models include the hierarchical model (data objects are represented by nodes called segments and connected by links in a tree-structured fashion), the network model (data objects are represented by nodes called records and connected by labeled links), and the relational data model (data objects are represented by flat tables called relations and defined by the Cartesian product of their attributes). The relational model has gained wide acceptance because of its simplicity and generality.

A *transaction* is a logically atomic set of actions (insertion, deletion, update) on occurrences of data objects of a database schema or subschema. A transaction is a unit of work which must be fully executed or not at all.

A *query* is a database operation involving retrieval of occurrences of some data objects under some conditions on data values.

Database schemata are structured by the database administrator (DBA) according to the procedures of some *logical database design methodology*, i.e., a structured, step-by-step method for building the database conceptual schema, its subschemata, and the sketch of major transactions. Two classes of database design methodologies can be distinguished: top-down methodologies which try to define the conceptual schema first and then derive the subschemata; and bottom-up methodologies which first model the subschemata or local views and then make the integration of subschemata to form the conceptual schema.

2.3.2. The Physical Level of Database Systems

Application programs and users access data through a complex software program, called the database management system. A *database management system (DBMS)* is a piece of software which manages databases and performs transactions on data. It is a software interface between external programs or users and data. Only the DBMS can have direct access to data. Therefore, there are three levels of abstraction in a database system: *the internal level* or database level, *the conceptual level* or DBMS level, and *the external level* or user level.

A DBMS is usually structured around one data model concept (hierarchical DBMS, network DBMS, or relational DBMS). According to this model, the DBMS provides three languages: a *data definition language (DDL)* for the use of the database administrator in his task of database schema definition and creation, a *data manipulation language (DML)* for the expression of transactions on data by on-line users or in the form of application programs, and a *query language (QL)* for the expression of interrogations on the database contents by on-line users or application programs.

The DBMS performs basic functions such as record insertions, deletions, updates, and retrievals plus more complex functions such as data security, data integrity control, concurrency management, and system recovery in case of failure. *Data security* means that the database contents can be protected against unauthorized access and from system malfunctions. *Data integrity* control is responsible for verifying that data inserted in the database are correct and that the database is in a consistent state. *Concurrency management* refers to the mechanism dealing with concurrent accesses to the database and resolution of conflicting accesses as do operating systems. *System recovery* is a mechanism for restarting the database after a crash of the system and which is able to restore the last consistent state of the database before the crash.

3. PRODUCTION SYSTEM ANALYSIS AND DATABASE DESIGN

3.1. Production System Analysis

Production system analysis or Factory analysis, i.e., making a complete modelling and analysis study of the enterprise, is one of the most important tasks in the process of automating a production system. It must also be one of the very first tasks of the automation project.

Managers must be provided with the best, most comprehensive, and clearest view of the current status of the enterprise in order to identify enterprise improvement possibilities, sketch new strategies, make what-if type simulations, and understand the effects of local changes on the overall organization of the company. Another advantage of factory analysis is that when the model of the production system has been built, it can be used and shared by people at the management level, at the engineering level, and at the manufacturing level so that enforcement of a common perception of the operations of the company can be achieved. Therefore, it provides a way of communication, standardization, and integration within the company. But the most important advantage of production system analysis is to give a global view of the current status of the production system, so that problems can be identified and the effects of their solutions on the existing system can be evaluated. However, in many automation projects, this phase is very often neglected and this lack of planning may be a cause of immediate failure as well as of long term problems.

It is now commonly accepted that CIM system planning and design must be supported by a structured modelling and analysis methodology involving an *AS-IS Analysis* task followed by a *TO-BE Conceptualization* task (13, 20).

The main objective of the AS-IS task is to provide management with a good model of the company (organization models, manufacturing or operation models, information models) in order to state what exists today and to make an accurate assessment of available capabilities. The model can also be useful to detect areas in which problems exist. From this modelling, opportunities and priorities for automation might be deduced.

The main objectives of the TO-BE task concern the definition of new (manufacturing) operations and the definition of the new information system, then the evaluation of the overall design, and finally the refinement of the solutions according to the real-world environment of the enterprise.

3.2. Database Design

Before a database can be implemented and used, it must be carefully designed. Since the quality of the database structure design will determine the success or failure of the data processing application project, some methods for supporting database designers in their work must be available. These methods are grouped into what is called a database design methodology.

Therefore, *database design* (6-8, 18, 19, 21) is the art of modelling the information structure of some enterprise, i.e., designing a canonical representation of operational data of the enterprise which balances all user and application data requirements for the purpose of automating some part of the enterprise.

Logical database design only deals with the definition of the general representation of the information structure, called the *conceptual schema*, with no considerations as to which physical database system it will be implemented on. Physical database design deals with the process of translating this neutral representation of the information structure into a database structure which can readily be implemented on a commercial system.

Database design is

- A nontrivial task which may last several months and require processing of many documents and involvement of many people
- A crucial task for the company or the organization since bad information modelling will lead to automation project failure
- A conceptualization task since the goal is to come up with an accurate model of the information structure of the company which can be shared by various people having different backgrounds

In the following, we give special attention to the Italian DATAID methodology which was developed as a 5-year project sponsored by the National Research Council of Italy and which involved cooperation of ten Italian Universities and Research Centres.

The DATAID methodology (22-25) follows the general methodological framework for database design proposed in refs. 2, 25 and 26. More specifically, the design phases are concerned with:

1. *Requirements Collection and Analysis*: Inputs to this phase are informal and heterogeneous descriptions of the observed organization. Three classes of descriptions are considered: natural language sentences (derived from documents, interviews, meetings,...), traditional or DBMS (Data Base Management System) files, and forms. Requirements are then analyzed to produce a collection of glossaries describing data (static representation), and operations and events (dynamic representation).
2. *Conceptual Design*: In this phase, a model (called conceptual local view) is built for each environment of the organization (view modelling). The design of the views follows an operational approach: for each operation in the environment, an operation schema is built which describes the required data. These schemata are then aggregated into one conceptual view, called the global view. In the view integration process, strategies (based on conflict analysis, merging, enrichment, and restructuring) are provided to deal with problems which may occur when objects (and their properties) which are common to different views have not been modelled in the same way. The local schemata and the global conceptual schema are formal representations of both static and dynamic data requirements of the organization.
3. *Logical Design*: This phase deals with translation of the conceptual schema into a logical schema for the target DBMS. Both relational and CODASYL-like DBMSs are considered in the DATAID methodology. This mapping process applies a set of transformation rules to the elements of the conceptual schema to obtain a solution which optimizes the execution of the most important operations of database applications.
4. *Physical Design*: The aim of this phase is to provide database designers with a complete set of design decision tools and techniques and to guide them through the process of selecting the best physical design for tuning the performance of the target DBMS. The main goal of the physical design phase, with respect to CODASYL-like databases, concerns the record placement strategy, the storage allocation design, and the access path supports. The main goal of the physical design phase for relational databases is the selection of an optimal set of secondary indices for each relation of the schema (it is assumed that the choice of the key has already been made during the logical design phase).

3.3. Methodological Framework for Implementing Production and CIM Databases

A methodological framework has been proposed for the development, implementation, and operational use of production and CIM databases (15). This methodology involves the following steps:

1. Specification of the production or CIM system (definition of a strategic plan)
2. Conceptual modelling and analysis of the system (using factory analysis methods)
3. Database design committee set up
4. Logical data modelling and logical database design
5. Selection of an appropriate physical database management system (DBMS)
6. Physical design and implementation of the database
7. Application program development
8. Database operational use and maintenance

The first step is performed by top management and consists of a clear statement of what in the company has to be automated. Then, the modelling and analysis step is directed towards a clear understanding of the actual state of the system and the implications of the automation project suggested by top management. Thus, the scope of application of the necessary database can be identified. Once this step is completed and

approval for the automation project has been issued, a database committee is formed, i.e., a group of people (from the company or from outside) is established to define the database system. The next step is the logical design of the database structure. Then, a commercial DBMS can be selected according to the specific requirements of the application project. The physical design of the database for this particular DBMS can now take place and the database can be implemented with this system. Application programs can be developed and coded by application programmers. Finally, operational use of the database can be started when application programs are completed and debugged.

What is new in this software development life cycle for CIM applications is the close relationship between production system analysis and modelling and database design. Production analysis and modelling methods constitute an extension of the data requirements formulation and analysis phase of database design methodologies. Also, they have the potential for supporting a good assessement of the future database system implications on the existing system to be automated. Moreover, system analysis and modelling methods can be used to precisely specify the scope of the database relative to the entire production system framework and these specifications can be expressed by means of simple, general tools which can be understood both by managers and by data analysts or database specialists. Therefore, integration of system analysis and modelling methods with database design tools in a common methodology for information system analysis of production systems is desirable.

4. THE M* GENERAL ARCHITECTURE

The M* methodology has been derived from the DATAID methodology for database design and from the expertise developed at NRC on CIM databases. The aim of the M* methodology is to provide engineers with design tools for the development of production and CIM databases.

Therefore, in accordance with the previous Section, the M* methodology covers the analysis of a production organization, the modelling of its information system, and the design of the related database environments which will support the automated operations of the organization. In fact, M* covers steps 2, 3, 4, and 6 of the methodological framework presented in Section 3.3.

With respect to production system environments, the M* methodology introduces some basic improvements and modifications to the DATAID methodology; more specifically:

1. A phase directed towards the precise analysis of the organization rather than just data requirements specification has been added as the first phase of the methodology to give a functional description of the organization and to specify the boundaries of use of the future database within the existing information system framework;
2. The representation of the information system dynamics has been modified with the introduction of a high level Petri net model;
3. The logical and physical design phases have been adapted to the DBS/R system, a relational DBMS developed at the National Research Council of Canada, which is used for the management of data (27).

As mentioned earlier, the M* methodology consists of three main design phases: (a) Enterprise Modelling and Analysis, (b) Conceptual Design, and (c) Implementation Design.

Enterprise modelling and analysis: This phase is used to analyze precisely the current status of the production system (AS-IS Analysis) by means of a top-down analysis (from management to production functions) followed by a bottom-up analysis (starting with the production function) and involving the users of the system to be modelled. Then, a TO-BE Analysis is performed which involves identification of problems of the production system, and specification of the new automated system. Both the AS-IS and TO-BE Analyses are specified by means of functional models (describing enterprise functions), information models (describing information objects of the enterprise), and dynamics models (describing time-dependent properties of enterprise functions). The scope of the database to be designed in order to support the new system is stated in the environment specification step which makes the transition with the next phase. A database environment is a subset of the production system consisting of highly inter-related users and activities involved in the same group of functions.

The *Conceptual Design* phase is used to analyze the functional description and to construct a conceptual description of the object enterprise in terms of two large classes of concepts related to static information and dynamic information.

Static information refers to the structure of data and to the integrity constraints that data must satisfy.

Dynamic information refers to the operations that must be performed on data and to the causal relationships existing among them.

For each environment, a local view is produced. These views are then integrated into one global view. A view is a complete description of both static and dynamic information, related to a specific environment or to the overall organization. A view contains a data schema to describe the data structure, a functional net, and a set of operation schemata to describe the organization behaviour.

The first two phases of the M* methodology are independent of the specific DBMS and software environment that will be used for the application development. Conversely, the subsequent phase is strictly related to the target system, DBS/R in our case.

The *Implementation Design* phase translates the global view into a data structure compatible with the target system. An implementation view (consisting of a relational schema, a physical schema, and a set of program fragments) is then produced.

Since the major aim of the M* methodology is database design and not information system design, it is supposed that other "ad hoc" methodologies (see, for instance, refs. 17 and 28) have been applied to deal with important aspects of organization analysis like the feasability study and the cost-benefit analysis.

The overall architecture of the M* methodology is illustrated in Fig. 2.

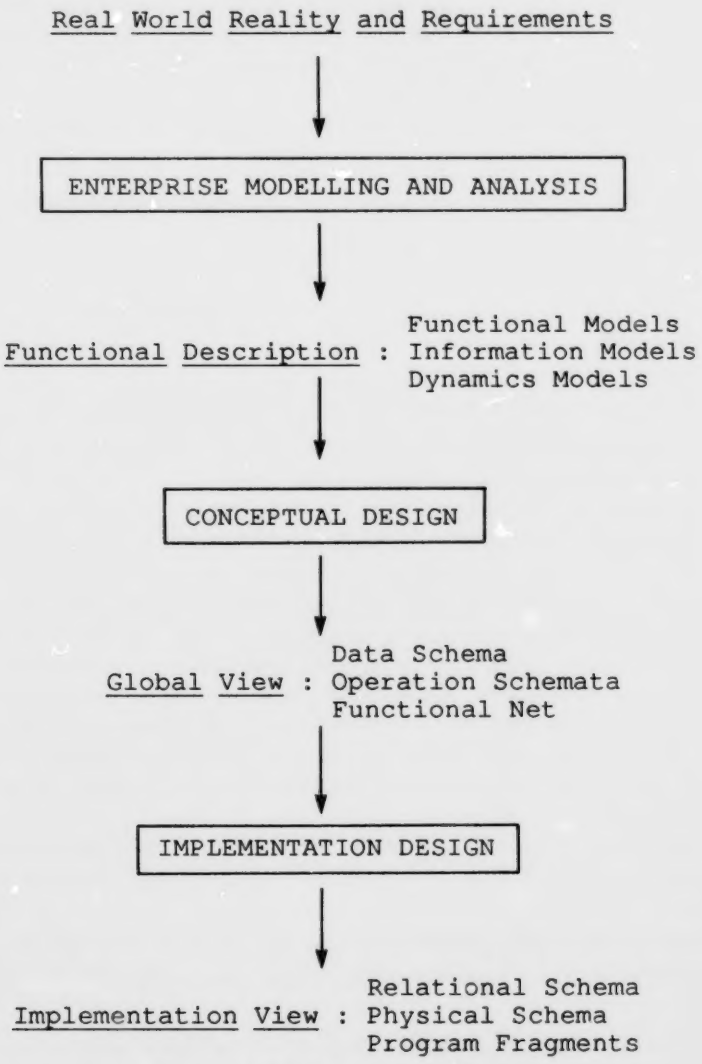


Figure 2: The M* Overall Architecture

5. THE INTEGRATED MODEL (OR M* TOOL-BOX)

5.1. Introduction

The fundamental aim of an Information System Design Methodology is to provide data analysts with a set of tools and methods to transform a user-oriented verbal description of a real-world object organization into a DBMS-oriented description, the database application (Fig. 3).

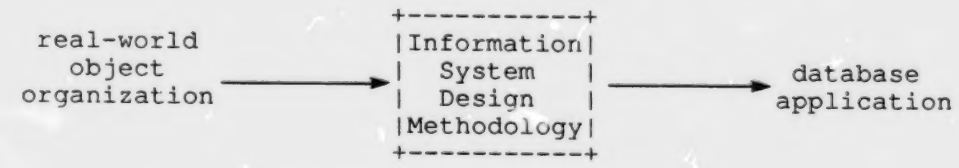


Figure 3: Information System Design Methodology

The *integrated model* of an Information System Design Methodology is the set of linguistic, descriptive, structural, and dynamic tools used at the different steps of such a transformation process.

The aim of this Section is to present all the basic tools or components of the M* tool-box, i.e., models, methods, and techniques, supporting the various phases of the M* methodology.

The integrated model on which the M* methodology is based is a "model type", i.e., it is mostly a set of descriptive models adapted to various groups of users of the information system. It allows the description of the object organization at three different levels: (a) *Organization Level* or management level for which a clear and accurate representation of the enterprise is required, (b) *Conceptual Level* or database system level which must be an interface level between managers and engineers and computer specialists, and (c) *Implementation Level* or application development level only of interest to computer specialists and application programmers. This classification is in accordance with the three methodological phases of the M* overall architecture of Fig. 2.

At the Organization level, a *Functional Model*, an *Information Model*, and a *Dynamics Model* are used by system analysts to provide management with a good and explicit description of the functions of the enterprise.

At the Conceptual level, a *Data Model*, an *Operation Model*, and a *Process Model* are used by database personnel to specify the structure of the data, the data manipulation operations, and the causal relationships among them.

At the Implementation level, *high-level computer languages* used for application developments are introduced.

5.2. The Organization Level

The models used at the Organization Level to describe the various functions and sub-functions of the object enterprise are aimed at helping in the specification of function inputs, function outputs, function behaviours, and relationships existing among them.

The Functional Model and the Dynamics Model of the M* methodology are essentially the same as the ones used in the IDEF methodology (11, 20). The information model has been changed to be more general at this level and to be completely independent of any computer data processing consideration.

5.2.1. The Functional Model

The Functional Model is based on well-known software engineering approaches using functional specification, such as SADT (Structured Analysis and Design Techniques) (16), Petri Nets (29), or DFD (Data Flow Diagrams) (17).

The Functional Model provides a description of the enterprise in terms of a hierarchy of functions. As mentioned earlier, a function is a set of activities to perform a task in the global operation of the enterprise or organization. In order to precisely specify a function, the modeller is required to identify what objects are input to or output from the function, what objects control or influence the function (control objects or constraints or conditions), and what is necessary to carry out the function (support objects such as pieces of equipment, tools, and people). The word object is used here in its broad sense and may represent everything (data, documents, orders, things, materials, ...). Output objects of some functions may be used as input objects or support objects or constraint objects of other functions.

In Table 1, the graphical symbols for the basic concepts introduced in the functional model are displayed. A function is represented by a rectangle (black-box representation). Input objects are represented by arrows pointing at the left side of the box, output objects are represented by arrows coming out of the right side of the box, control objects or constraints are represented by arrows pointing at the top side of the box, and support objects are represented by arrows pointing at the bottom side of the box. Objects are represented by their names.

Usually, in most real-world situations, it is not possible to obtain directly the overall representation of the organization at a given level of detail. Moreover, different environments may require different levels of detail.

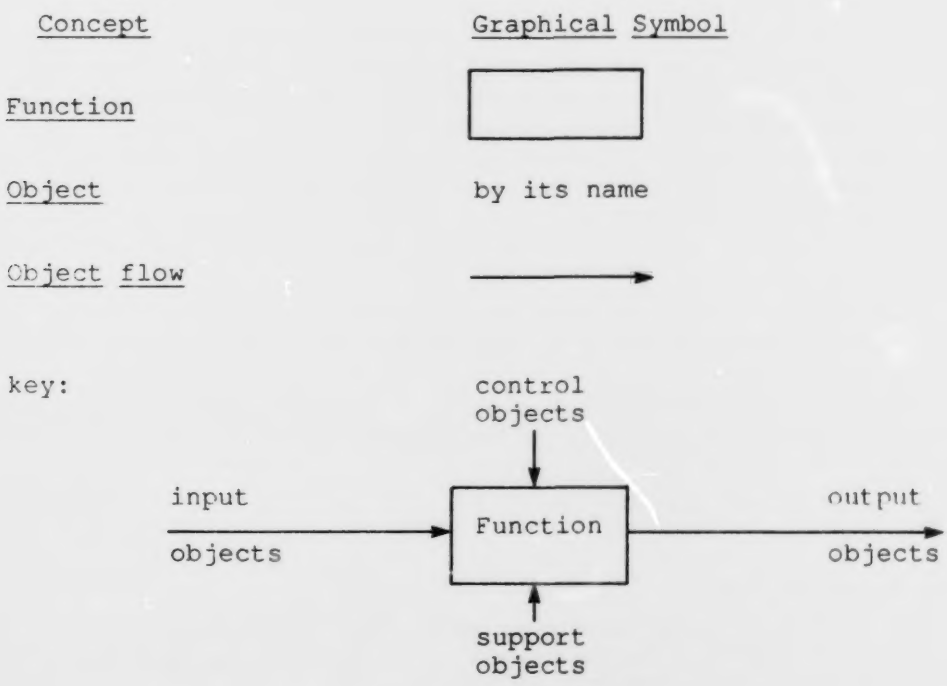


Table 1: Functional Model Formalism

The functional decomposition is a widely adopted technique to face the problem of managing real-world complexity. Decomposition (or top-down refinement) is a technique used to describe a function at the nth level in the hierarchy of functions in terms of sub-functions at the (n-1)th level. For example, in Fig. 4, the general functional diagram of a function is given at the nth level in terms of sub-functions A and B. Sub-function B is then broken down at the (n-1)th level in terms of sub-sub-functions B1, B2, and B3.

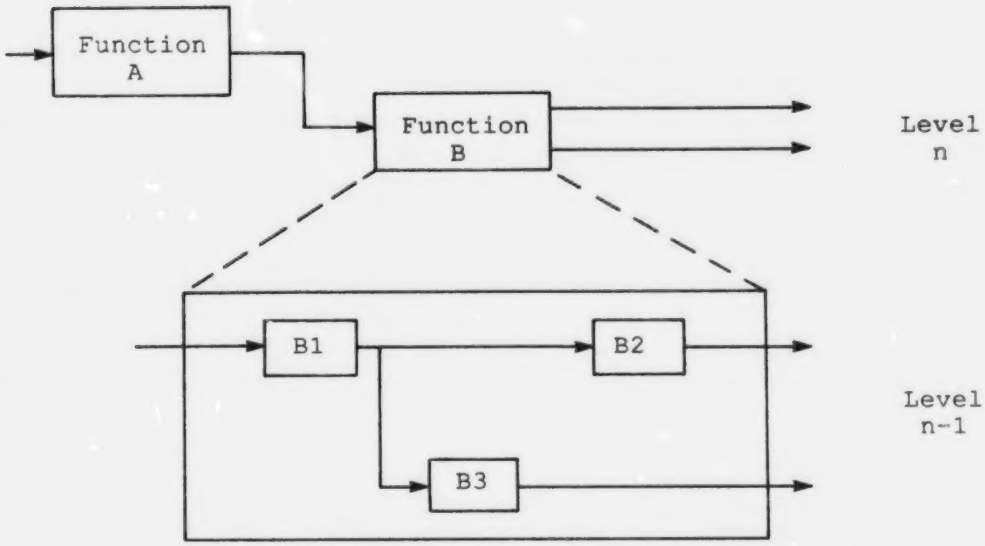


Figure 4: Example of Function Decomposition

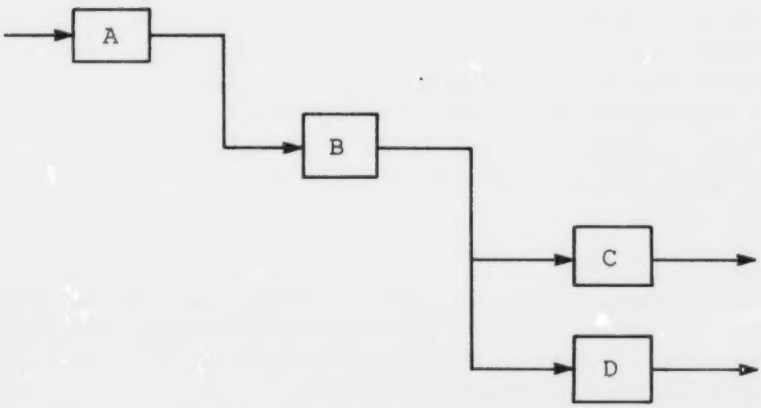


Figure 5: Example of Chronology of Functions in a Functional Diagram

The chronology of execution of the functions may also be of importance and can be represented. In this case, functions which must be executed before others are represented on the left of these functions in the functional diagram. If the execution of a function is dependent on the execution of another function, the dependent function rectangle will be placed on the right and below the other function rectangle. For example, in Fig. 5, function A must be executed before function B which is executed before functions C and D while functions C and D are executed in parallel.

5.2.2. The Information Model

The Information Model provides a description of the information circulating within the enterprise. It only deals with the analysis of information objects which appear in the functional model and constitute the interfaces among the organization functions. The purpose of this analysis is to classify information objects

according to the following classes:

- *Data*: Data are information items relevant to the organization, i.e., they are pieces of information used by the functions within the organization environments;
- *Messages*: A message is a piece of information concerning the fact that something has happened. According to this definition, a message is the organization concept related to the fact that an *event* can happen in the real-world (e.g., it is the end of the month, an order has arrived, etc.);
- *Files*: A data file is a structured set of records composed of data-items. Records represent real-world objects as manipulated by computers.
- *Forms*: A form is a document that is used to collect, display, or communicate data within the organization environments.

The graphical representation used for the basic concepts previously introduced in the model is displayed in Table 2. When the nature of an information object is not obvious to determine, it is classified as a data element.

5.2.3. The Dynamics Model

The Dynamics Model is used in the M* methodology to analyze the behaviour of an organization function when time is an important characteristic in the description of the function. This model represents the time dependent features of functions and objects interacting over time.

In addition to the concept of function, the basic concepts described with this model are *queues* and *precedence relationships*. A queue represents the fact that a set of homogeneous objects are waiting for processing by a certain function. A precedence relationship defines an ordering in the execution of two correlated functions. Moreover, the model also takes into account the notion of support of a function, i.e., external intervention during the time-dependent execution of the function.

The graphical representation used in the Dynamics Model is illustrated in Table 3. Note that in this formalism, a function is simply represented by an arrow with the function name above.

To illustrate the various principles of the integrated model, the following very simple example adapted from Dijkstra's famous problem about philosophers will be considered:

The Dining Philosophers problem:

Five philosophers are seated at a round table, eating spaghetti. Between each philosopher is one fork. To eat, a philosopher must pick up both the fork on his left and the fork on his right. When a philosopher is not eating, he is thinking and vice versa. The problem is to synchronize the activities of philosophers since they cannot all eat at the same time because of lack of forks.

The system can be easily modelled by means of a single function called "Table Management" (Fig. 6).

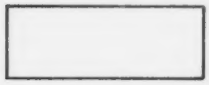
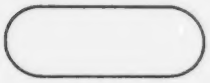
5.3. The Conceptual Level

At the Conceptual Level, a complete formal description of both static and dynamic aspects of the object organization must be produced in terms of information management.

As mentioned previously, the Conceptual Model consists of three models: (a) a *data model*, (b) an *operation model*, and (c) a *process model*. In this paragraph, the graphical representations used at the conceptual level will be illustrated.

5.3.1. The Data Model

The Extended Entity Relationship (EER) model (22, 23) is used in the M* methodology to describe the information objects of interest for the organization. The EER model derives from the entity-relationship

Concept	Graphical Symbol
Function	
Information Object:	
- data	
- message	
- files	
- form	
Object flow	

key:

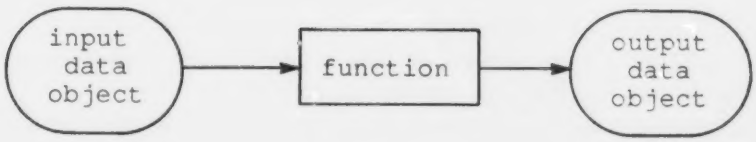


Table 2: Information Model Formalism

(ER) model proposed in ref. 30. It provides the designer with four abstraction mechanisms: entities, relationships, attributes, and abstraction hierarchies.

1. *Entities*: An entity describes a set of objects (entity occurrences) which share common characteristics.
2. *Relationships*: A relationship R among the entities E1, E2, ... describes the set of relations (relationship occurrences or *links*) relating the objects e1 $\in$ E1, e2 $\in$ E2, ..., and (possibly) sharing common characteristics.
3. *Attributes*: An attribute describes a characteristic or property of an entity or a relationship, and takes values from a value set called the *domain*.
4. *Abstraction hierarchies*: The first type of hierarchy is the *subset hierarchy*. An entity S is a subset of another entity E (E is a superset of S) if every occurrence of S is also an occurrence of E. The second type of hierarchy is the *partition hierarchy*. Entities P1, P2, ... are a partition of the entity E (E is a

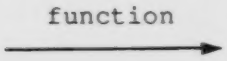
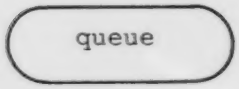
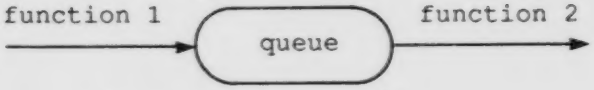
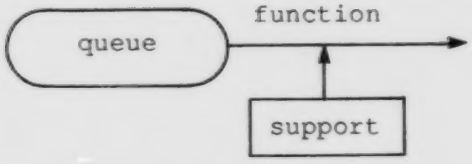
Concept	Graphical Symbol
Function	
Queue	
Precedence relationship	
Support	
key:	

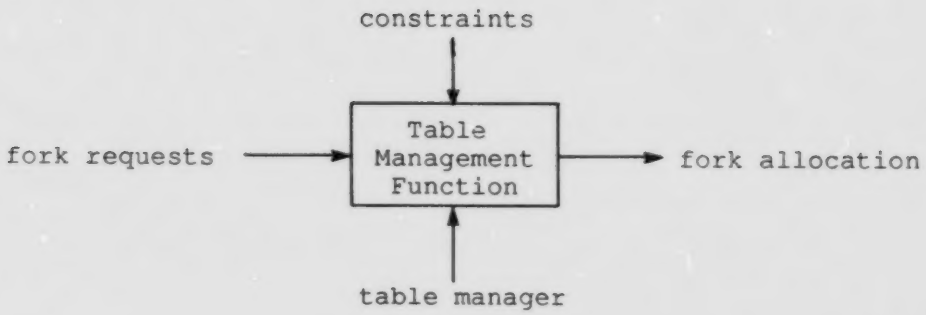
Table 3: Dynamics Model Formalism

generalization of P1, P2, ...) if each occurrence of E is also an occurrence of one (and only one) of the entities P1, P2, ... , . The partition on the occurrences of E can be explicitly represented by an attribute of E. In this case, it will be referred to as a discriminating attribute.

The EER model allows the definition of a wide range of *integrity constraints*, i.e., rules defining properties of data. Among them, the following are especially relevant:

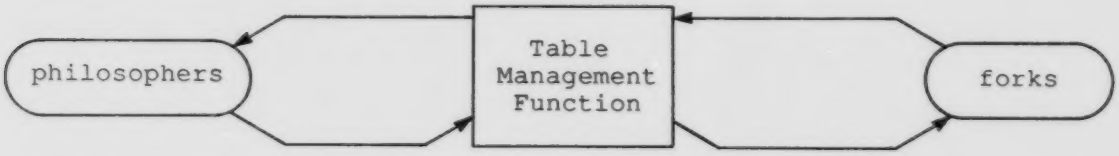
1. Entity constraints: Entities must be uniquely determined by means of (at least) one *identifier*. An internal identifier is an attribute (or group of attributes); entities may also be identified by means of other entities associated with them (external identifiers).
2. Relationship constraints: They are related to the mapping functionalities of a relationship. These functionalities, e.g., 1:1, 1:N, N:M, and total/partial nature, can be expressed by means of two natural integers: the min- and max-cardinality (31). The min-cardinality is the minimum number of times (possibly 0) that each object (entity occurrence) can be involved in a link (relationship occurrence). The max-cardinality is the maximum number of times (possibly 1) that each object can be involved in a link.
3. Attribute constraints: An attribute can be *simple* (the corresponding domain is atomic), *repeating* (it can take several values), *aggregated* (when it is composed of several simple and/or aggregate attributes). Moreover, an attribute can be *total* (the value must always be specified) or *partial* (during the lifetime of the entity, the attribute may be unknown).
4. Hierarchy constraints: Both in subset and partition hierarchies, attributes and relationships of the entity E (at the upper level) are attributes and relationships of the entities at the lower level, but not vice versa. Usually, entities at the lower level have additional attributes that specify particular features of

Functional model:



constraints:
- to eat, a philosopher must pick up both the fork on his left and the fork on his right.
- a philosopher is either thinking or eating

Information model:



Dynamics model:

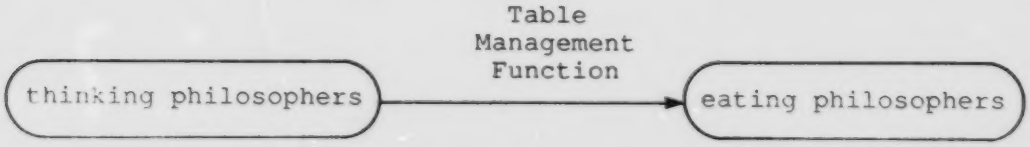


Figure 6: The Dining Philosophers System

these entities with respect to the entity at the upper level.

The graphical symbols for the EER model abstraction mechanisms are shown in Table 4.

The data schema for the Dining Philosophers problem expressed in terms of the EER model is shown in Fig. 7. The relationship P_F associates, at a given time, each eating PHILosopher with the FORKs he uses.

5.3.2. The Operation Model

At the Conceptual Level, an operation is described by means of a data schema expressed with the formalism of the EER model, called the *operation schema*. The goal of this schema is to specify the names and the types of the data needed by the operation and to ensure that these data are consistent with the overall data schema.

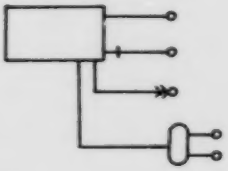
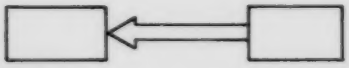
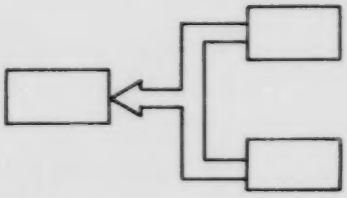
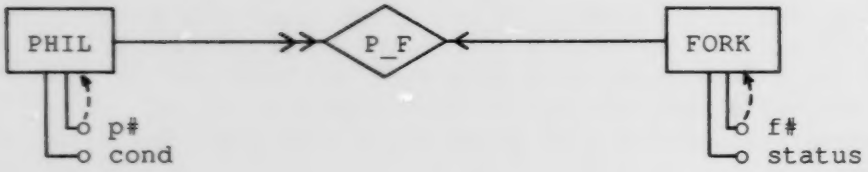
CONCEPT	REPRESENTATION
entity:	
relationship:	
total/partial	
1:1 / 1:n / n:m	
attribute:	
total/partial	
repeating	
aggregate	
identifier:	
internal/external	
subset hierarchy:	
partition hierarchy:	

Table 4: EER Model Formalism

In the Dining Philosophers example, two operations can be considered:

1. pick_up forks (both the fork on the left and the fork on the right)
2. put_down forks

From the database point of view, pick_up means that the state of the database must be modified by updating both the attributes "cond" (from "thinking" to "eating") of the object PHILosopher and "state" (from "unused" to "used") of the corresponding FORKs, and by inserting two links (occurrences of the relationship P_F) connecting these objects. The put_down operation performs exactly symmetric actions. In both cases, the operation schema coincides with the data schema of the database as presented in Fig. 7.




```
domains      p# : (1..5)
           cond : (thinking, eating)
             f# : (1..5)
         status : (used, unused)
```

Figure 7: The Dining Philosophers Data Schema

5.3.3. The Process Model

Process modelling is used to represent the organization behaviour, that is, the possible sequences of operations of the organization functions within a given environment, and the communications among them.

In the M* methodology, the function behaviour is described by means of a *process net*, that is, the set of operations with their associated execution conditions, which must be executed to achieve the aims of the object function.

Execution conditions are usually related to messages coming from the external world (with respect to the object function) and/or to the existence of some objects in a particular state (message and data driven systems). In the M* approach, both messages and objects are stored in the information system database; thus, they can be checked simply by looking into the database. Moreover, in general not all possible combinations of the objects corresponding to the input conditions can "fire" the operation; usually, these objects must satisfy some constraints (or input rules).

When an operation has been executed, the state of the system is changed; thus, modifications in the information system state must be recorded in the database.

In this way, checking of execution conditions corresponds to database queries (or simply queries), and real life operations correspond to database update actions.

For each function, a process can be modelled using a *process net model* (32).

The process net model, which is derived from the well-known Petri Net model (29), integrates the process description capabilities of the Petri Net model with the data description capabilities of the EER model.

Formally, a process net related to the database S consists of:

1. A bipartite directed graph (P, T, F) where P and T are two disjoint sets of vertices ($P \cap T = \emptyset$) called *Conditions*, and *Operations*, respectively, and F is a set of directed arcs, each one connecting a condition $p \in P$ to an operation $t \in T$ or vice versa ($F = P \times T \cup T \times P$). In the original terminology on Petri Nets, P and T are called Places and Transitions, respectively.
2. A set {Q(p)} of *queries* on the database S, associated with the set $P \subseteq \{p\}$ of conditions.

   A set {U(t)} of *transactions* on the database S, associated with the set $T \subseteq \{t\}$ of operations and the corresponding rules.

An operation $t \in T$ is *enabled* whenever:

a. For each input condition p of t, it corresponds to a result of the query Q(p) that is not empty.

b. There exists at least one set of objects, extracted from the input conditions, for which the rules specified in the transaction U(t) are satisfied.

When an operation t is enabled, it can be *fired*. From the database point of view, this means that the update actions defined in the transaction U(t) associated with the operation t have to be applied. Update actions modify the database state and impact on the output conditions of t. Also the content of the query results associated with the input conditions of t are usually modified.

In the Dining Philosophers problem, rules and input conditions for the operations pick_up and put_down are simply:

1. pick_up:

   -input conditions: P (correspond to philosophers who are thinking), F (correspond to forks which are unused);

   -rules: the unused forks must be those on the left and on the right of the object philosopher.

2. put_down:

   -input conditions: E (correspond to philosophers who are eating);

   -rules: (this operation has no particular rules).

The complete description of the dynamic behaviour of the system is shown in Fig. 8. In this figure, the operation schemata corresponding to the input conditions are also reported (the operation schemata corresponding to the operations pick_up and put_down coincide with the database schema).

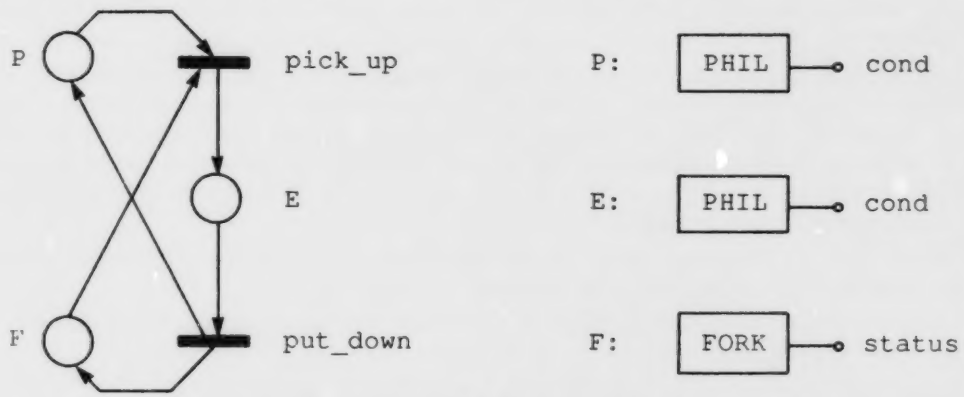


Figure 8: The Dining Philosophers Process Net

As it can easily be seen, the process net model presents the same characteristics as the Predicate Transition Net (PrTN) model introduced in ref. 33. The formal definition of the process net model and its relation with the PrTN model is developed in ref. 28. The most important feature of the process net model, with respect to the PrTN model, consists of the explicit connection with the data description and the operations specifications. This connection plays a crucial role in the modelling phase of the information system life cycle.

5.4. The Implementation Level

The Implementation Level corresponds to the physical DBMS which is used to develop the database application, i.e., the DBS/R relational system in this report.

At this level, three languages are used to describe static and dynamic features of the object organization: (a) a *data definition language*, (b) a *data manipulation language*, and (c) a *process definition language*.

5.4.1. The Data Definition Language (DDL)

The DDL is a relational definition language which is defined at length in ref. 27. The relational model supported by the DBS/R system is an extension of the conventional relational model (6) because it allows the definition of non-simple attributes: (a) array attributes (ARRAY), and/or (b) structured attributes (RECORD). These types of attributes can be used to model repeating and aggregate attributes of the EER model.

The DDL will be briefly illustrated with the Dining Philosophers problem. The relational database corresponding to the EER database schema consists of three relations. These are shown in Fig. 9.

```
PHIL(p#, cond); P_F(p#, f#); FORK(f#, status)

which are specified by means of the given statements:

TYPE
 Tcode = interval: 1..5;
 Tcond = (thinking, eating);
 Tstat = (used, unused);
 Tphil = RECORD
           p#    : Tcode;
           cond  : Tcond
         END;
 Tfork = RECORD
           f#      : Tcode;
           status  : Tstat
         END;
 Tlink = RECORD
           p#, f# : Tcode
         END;

RELATIONS
 PHIL : Tphil; KEYS: p#       ; ESTIMATED POPULATION: 5;
 FORK : Tfork; KEYS: f#       ; ESTIMATED POPULATION: 5;
 P_F  : Tlink; KEYS: (p#, f#); ESTIMATED POPULATION: 4;
END
```

Figure 9: The Dining Philosophers Relational Schema for DBS/R

Data definition statements are then compiled and used to create the related database management system (DBMS) for this database schema.

5.4.2. The Data Manipulation Language

In the remainder of this Section, a simple High-Level Interface (HLI) for the DBS/R system will be briefly described.

The HLI is a navigational oriented interface obtained by embedding a relational Data Manipulation Language in a host programming language. The translation between HLI programs and DBS/R programs is quite simple and can be done by means of an application program pre-processor of the DBS/R system.

The Data Manipulation Language of the HLI is derived from the QUEL language (34) with some simple extensions to take into account the extended attribute facility of DBS/R.

In general, queries in QUEL have the following form:

```
range of <var> is <rel>
{range of <var> is <rel>}
retrieve [into <rel>] ([<att>=]<var>.<field>
         {, [<att>=]<var>.<field>})
where <pred>
```

where brace brackets {...} denote repetition, and square brackets [...] denote optional items; <rel> refers to a relation; <att> is an attribute name; <var> is a tuple variable which can be instantiated to reference any tuple of the given relation; and <field> can be a simple attribute or an element of a non-simple attribute. Predicates are Boolean expressions involving comparision of attribute values; thus, grammatical production rules to express predicates in QUEL may be:

```
<pred> ::= <pred> "and" <comp> | <pred> "or" <comp> |
           "not" <pred> | <comp>
<comp> ::= <val> <op> <val>
<op>   ::= "=" | "-=" | ">" | ">=" | "=<" | "<"
<val>  ::= <var>.<field> | <constant>
```

QUEL also provides delete, replace, and append statements to manage relations and to perform set operations. In the example, the operations involved in the Dining Philosophers problem are described by means of the statements shown in Fig. 10.

5.4.3. The Process Definition Language

The QUEL language can be used as a data manipulation sublanguage embedded in a host programming language (in the DBS/R case, the PASCAL language). A precompiler parses the QUEL statements and replaces them with valid PASCAL calls to run-time modules that perform the desired functions.

A query:

range ... retrieve (x1 = v.f1, x2 = v.f2, ...) where ...

may appear anywhere in a program. The variables x1, x2, ... must be program variables that are used as targets for the values of the selected attributes v.f1, v.f2, ... It must be noted that if the query selects a set of tuples, just the first one is returned to the program.

In general, the user may wish to identify a set of tuples and process them one at a time. For this purpose, two basic concepts are introduced: (a) *navigation* in a window, and (b) *order* of retrieval (6).

A window is associated with a query by means of a declarative statement:

windows

W : range ... retrieve (x1 = v.f1, x2 = v.f2, ...) where ...

where W is the window name. At any point in the program, the window could be opened by means of a statement that activates the window (executes the corresponding query) and associates a *cursor* with the resulting relation. The cursor is a pointer that can be used to retrieve the resulting tuples, one at a time, by means of a *fetch* statement. Moreover, an *order_by* clause can be used to order the query result according to specified attributes. For instance:

```
P: range of x is PHIL
   retrieve into P (p# = x.p#) where x.cond = thinking

F: range of y is FORK
   retrieve into F (f# = y.f#) where y.status = unused

E: range of z is PHIL
   retrieve into E (p# = z.p#) where z.cond = eating

pick_up : range of p is P
          range of f1 is F
          range of f2 is F
          replace PHIL (cond = eating) where PHIL.p# = p.p#
          replace FORK (status = used)
              where FORK.f# = f1.f# and FORK.f# = p.p#
          replace FORK (status = used)
              where FORK.f# = f2.f# and FORK.f# = ((p.p# - 1) mod 5)
          append to P_F (p# = p.p#, f# = f1.f#)
          append to P_F (p# = p.p#, f# = f2.f#)

put_down: range of e is E
          replace PHIL (cond = thinking) where PHIL.p# = e.p#
          replace FORK (status = unused) where FORK.f# = e.p#
          replace FORK (status = unused)
              where FORK.f# = ((e.p# - 1) mod 5)
          range of u is P_F
          delete u where u.p# = e.p#

Note: pick_up and put_down are database transactions, i.e., they  must
be executed completely or not at all.
```

Figure 10: The Dining Philosophers Operations

```
open c1 on W order_by x1;
  while -eof c1 do
    begin
      ...
    fetch c1;
      ...
    end
```

is a piece of program code which activates the window W and opens an ordered cursor c1 on W; c1 is then used to retrieve the selected tuples one tuple at a time by the fetch command of the window in the specified order. Figure 11 shows the DBS/R PASCAL program for the Dining Philosophers example.

5.5. Conclusion

This closes the Section on the various tools forming the integrated model of the M* methodology. Table 5 gives a summary of the tools presented as well as the level at which they can be applied and the users they are supposed to help.

In the following Section, the case study is presented. Then, in the subsequent Sections, all phases of the methodology are successively considered and described with reference to the relevant tools to use. Examples taken from the case study demonstrate the use of these tools.

```
VAR
  x1, y1, z1: INTEGER;

 ...
 ...

WINDOWS
 P : ...
 F : ...
 E : ...
 ....
 ....
 open c1 on P;       /* the relation P is created */

   fetch c1;                /* now x1 contains the p# of the
                              first "thinking" philosopher */
      ...

 open c2 on F;       /* the relation F is created */
      ...
 pick_up: ...
      ...
 open c3 on E;       /* the relation E is created */

   fetch c3;                /* now z1 contains the p# of the
                                       first "eating" philosopher */
      ...

 put_down: ...
      ...

Note: The logic of the program must take care to avoid infinite loops
      (pick_up, ..., put_down, ..., pick_up, ...).
```

Figure 11: The Dining Philosophers Program

Level	Users	Tools
Organization	Managers	Functional model
	Engineers	Information model
	Analysts	Dynamics model
Conceptual	Engineers	Data model (EER model)
	Analysts	Operation model
	DB personnel	Process net model
Implementation	DB personnel	Data Definition Language
	Programmers	Data Manipulation Language
	DB users	Process Definition Language

Table 5: The M* Tool-Box

6. THE CASE STUDY

In this Section, a hypothetical company, called ABC, is introduced. Throughout the remainder of this document, we will refer to this example for illustration of the various phases of the M* methodology which will be considered in turn in the following Sections.

Let us assume that the ABC company produces turbine blades on customer demand with a Flexible Manufacturing System (FMS) environment involving flexible machining cells (FMCs) composed of a number of machine-tools (including NC-machines, robots, measuring machines, ...), some clamping devices and pallets, cutting tools, and measuring tools and devices. The production environment considered here is based on a real FMS developed at the Manufacturing Technology Centre of the National Research Council of Canada.

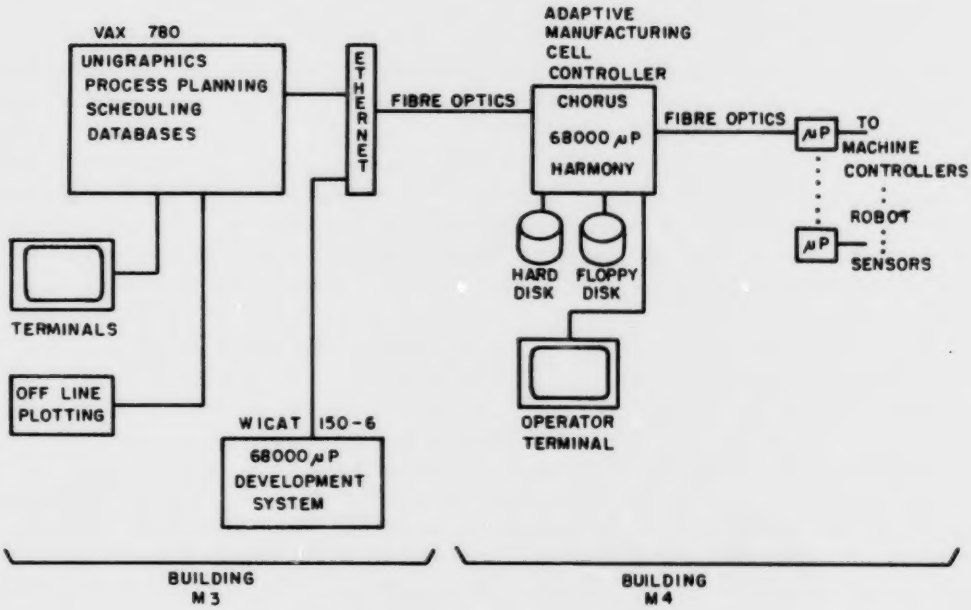
The aim of a FMC is to manufacture fabrication lots according to some predefined schedules. A fabrication lot is a batch of similar parts to be produced. Each occurrence of a part of the same lot is a workpiece made of some material. The workpieces are manufactured according to a predefined sequence of machining operations specified by a process plan. The process plan indicates the sequence of operations to be performed for a given part. It also indicates which machine to use with which tool type for each operation. Therefore, it is necessary to store the kinds of operations each machine can perform and the kinds of operations for which each tool type can be used. Finally, for a given part, standard time information per possible operation (such as setup time, labor time, queue time) must be recorded for subsequent analysis.

In this example, NC programs are not consider as real objects but just as manufacturing resources. Thus, only the references to the files containing them are stored in the database.

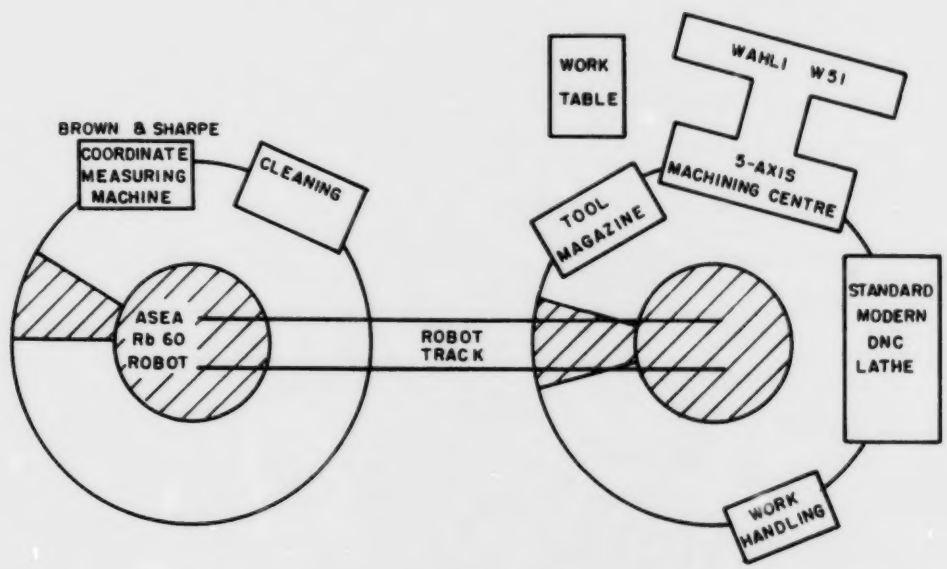
The FMC currently in place at the Manufacturing Technology Centre includes:

- a 5-axis machining center (WHALI);
- a CNC lathe (LATHE);
- a coordinate measuring machine (CMM);
- an industrial robot (ASEA);
- a tool coupler (TOOL);
- a control computer with a communication network.

A schematic representation of the NRC's FMC is shown in Fig. 12.



DME CAD/CAM FACILITY AND MANUFACTURING CELL



ADAPTIVE MANUFACTURING CELL

Figure 12: The Flexible Manufacturing Cell of NRC

7. Phase 1: ENTERPRISE MODELLING AND ANALYSIS

7.1. Introduction

The Enterprise Modelling and Analysis phase of the M* methodology is aimed at supporting a fine structural analysis of the enterprise in terms of the AS-IS Analysis and TO-BE Analysis as presented for production system analysis in Section 3.1. The goal is to get a clear understanding of the functions and operations of the enterprise expressed in terms of the functional model, the information model, and the dynamics model of the organization level of the M* integrated model (see Section 5.2). Then, an Environment Specification task is performed in order to define the limits of the database application to be designed. Based on results of the previous analysis tasks, the Environment Specification task involves the definition of a database committee, establishment of a data requirements collection plan, and the collection of these requirements from enterprise users and functions. Requirements will be classified into formatted requirements and nonformatted requirements. The overall architecture of the Enterprise Modelling and Analysis phase is shown in Fig. 13.

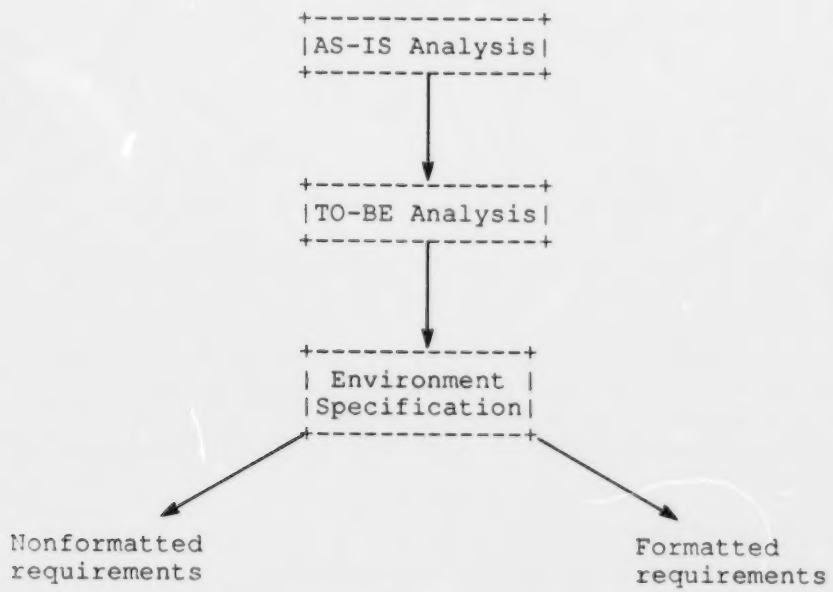


Figure 13: Enterprise Modelling and Analysis Phase Overall Architecture

7.2. Task 1.1: AS-IS Analysis

The aim of the AS-IS Analysis task is to provide managers and engineers with an accurate model of the enterprise on which they can make a good assessement of the current status of the enterprise. The overall architecture of the AS-IS Analysis task is shown in Fig. 14. In the AS-IS Analysis, we distinguish two steps: top-down analysis and bottom-up analysis.

7.2.1. Task 1.1: Input Description

The input of the AS-IS Analysis task consists mostly of interviews with managers, engineers, foremen or crew leaders, and users or workers.

Top-down analysis is performed on the basis of manager interviews only and especially interviews involving top managers and decision-makers. Other inputs to this activity are the enterprise organization chart, if it

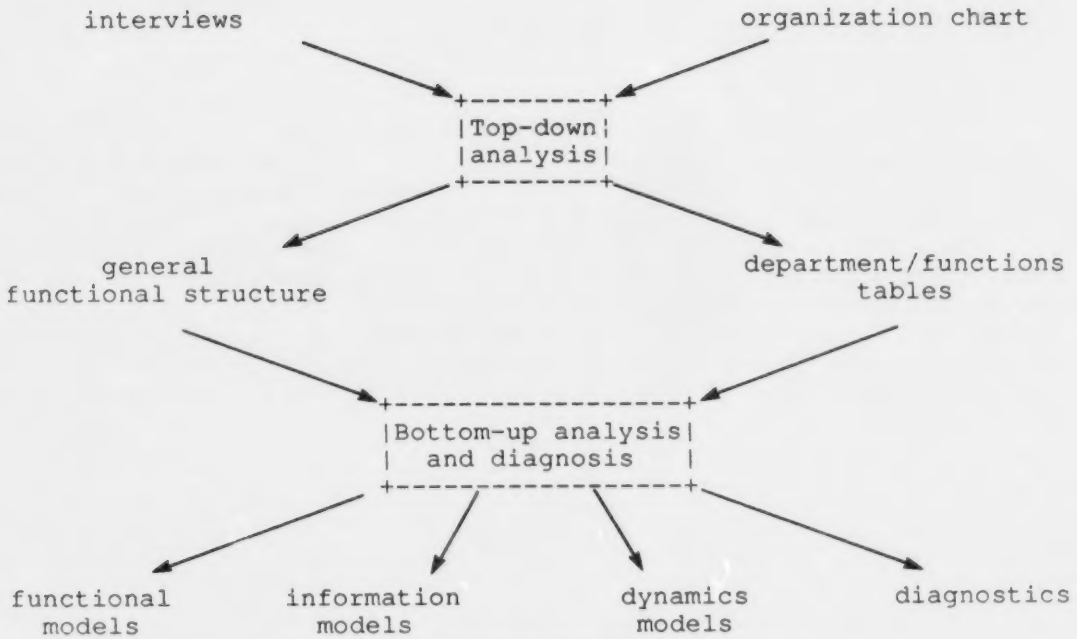


Figure 14: AS-IS Analysis Task Overall Architecture

exists, and documentation brochures in the case of companies. These documents give information about the gross organizational structure of the enterprise in terms of departments, managers, number of workers per departments, kind of service produced, etc.

Bottom-up analysis is performed by means of interviews involving department managers but also supervisors, foremen, and workers. People are asked to describe the functions in which they are involved and the activities that they perform. Documents processed are identified and the flow of documents modelled. Verbal description of the processing of documents or of the execution of function activities are recorded on tapes and reported in written notes.

7.2.2. Task 1.1: Output Description

The output of the AS-IS Analysis task consists of a report containing:

- A general functional structure of the enterprise and eventually a cross-reference table between departments and functions for the top-down analysis activity
- Functional models, information models, and dynamics models describing the static and dynamics structure of the enterprise and function diagnostics for the bottom-up analysis activity

The AS-IS Analysis report will be submitted to managers. It will be used to make decision about automation or computerization opportunities.

7.2.3. Task 1.1: Description of Activities

The procedural description of the activities of the AS-IS Analysis task is shown in Fig. 15.

The top-down analysis usually requires two to three short meetings with enterprise directors (2 hours maximum each). The discussions will concern the general organization of the enterprise, the size of the enterprise, the financial situation, the general problems faced by the enterprise, etc. One meeting can be used to present the analysis tools and approach to management.

The bottom-up analysis requires at least three meetings per enterprise function: one to make the interview and the function diagnosis with the director or supervisor (1 to 2 hours), one to model the function, i.e., identify inputs, outputs, and constraints and operations (2 hours or more), and a short one to discuss the results obtained with function personnel (1 hour). However, since this task can be highly iterative, more meetings (or longer meetings) may be required depending on the complexity of the analysis to be performed.

A detailed description of these analysis activities follows.

```
[top-down analysis]
1. meet the enterprise directors
2. consult the organization chart
3. identify all departments, their managers,
   and department relationships
4. produce the general functional structure
5. produce the department/functions table
6. plan bottom-up analysis meetings

[bottom-up analysis and diagnosis]
7. analyze enterprise functions starting from
   production/service functions

for each function
   7.1 interview supervisors and users
   7.2 make the function diagnostic
   7.3 identify function/sub-function inputs, outputs,
       constraints, and supports
   7.4 build the functional model, the information model,
       and the dynamics model
   7.5 validate models

end for

8. Check consistency and completeness of the global
   enterprise models
9. write the AS-IS Analysis report
```

Figure 15: AS-IS Analysis Activities

7.2.4. Top-down Analysis

The goal of this step is to understand the overall structure of the company, i.e., to discover the pyramidal structure of its decision system, identify the various levels of decision-making, and identify the decision-making centers. This step is crucial when the analyst comes from outside the enterprise to be analyzed. In general, it is recommended that the analysis team should not be too familiar with the enterprise activities to be analyzed.

Usually, to perform this task, the analysis team meets the directors and/or managers of the enterprise and eventually the supervisors of the departments. The tools used at this step are interviews and consultation of the organizational chart of the company. The outputs are: (a) a general functional diagram (indicating the general functions involved in the company with the names of their directors, and their relationships in terms of information exchange), and (b) eventually a cross-reference table between departments and functions if several departments are responsible for various distinct functions. Figure 16 gives an example of general functional diagram for the hypothetical company ABC producing turbine blades.

From this general understanding of the enterprise, a plan can be developed for the bottom-up phase, i.e., deciding from where to start the analysis and whom to interview.

7.2.5. Bottom-up Analysis and Diagnosis

The goals of this step are (a) to validate the top-down analysis, i.e., the general functional diagram, (b) to finely model each function identified in the general functional diagram, and (c) to make a diagnosis of each function in order to evaluate its operations. Each function can be broken down into more detailed sub-functions and each sub-function can in turn be decomposed if necessary.

The method that we have adopted to perform this step is:

1. Start from the bottom level of the structure and go toward the top, e.g., for a manufacturing company, we start at the manufacturing function.
2. For each function, we perform an interview/modelling/validation/integration analysis cycle, i.e., we make a diagnosis of the function by means of interviews of the users, workers, or operators; then, we model each function in terms of its inputs, outputs, information flow, dynamicity, and sub-functions; next, the modelling is validated with the help of the people involved in the function and the sub-functions; finally, the various models of the functions are verified and compared to one another for consistency and completeness (see Fig. 15).

Each function and sub-function is described by:

- a general description (in natural language)
- a functional model
- an information model
- a dynamics model

The general description of the function or sub-function is just a narrative description. The functional model, the information model, and the dynamics model principles have been explained in Section 5.2. These various tools are illustrated in Fig. 17 for the Engineering function of the company ABC (involving CAD, Process Planning, and NC Programming sub-functions).

The diagnosis is based on a method by Boyer et al. (35). It is a step-by-step method for guiding interviews with the users. The method indicates the kind of questions the users should be asked to evaluate the actual operation of functions commonly found in production systems (such as manufacturing, scheduling, process control, shop floor control, inventory control). Interviews are usually taped for subsequent reference and diagnosis results are reported in a written document.

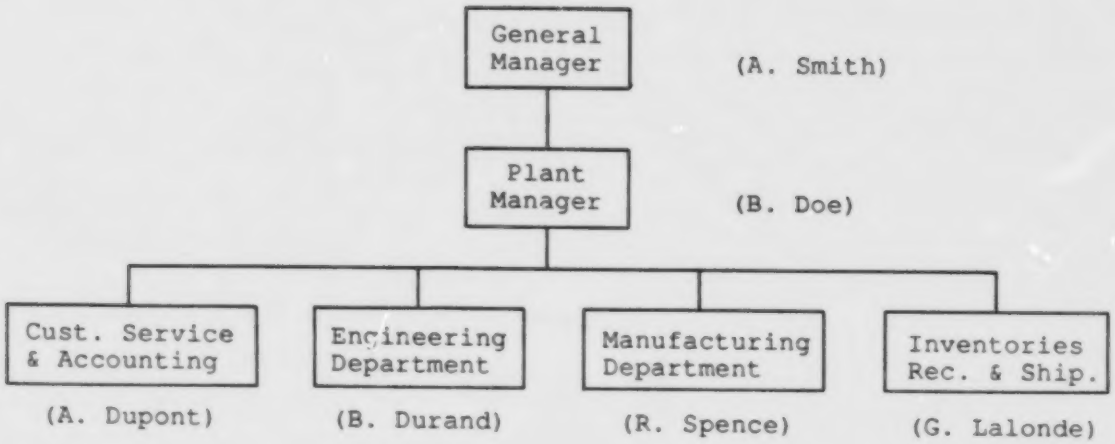
7.3. Task 1.2: TO-BE Analysis

Once the AS-IS Analysis task has been completed, the overall structure and operations of the enterprise must be evaluated. On one hand, the effort of analysts will be directed towards identification of anomalies or malfunctions in the operations of the system. On the other hand, managers will try to identify opportunities for automation or modernization. Then, proposals will be made to restructure the existing system into a new, more productive or more efficient system. One proposal will have to be selected and approved by management. Finally, the new system will have to be designed on paper.

In this report, we deal only with automation projects involving a database for their implementation. Therefore, the TO-BE Analysis task as described in this document is biased from an information system viewpoint. In the M* methodology, the TO-BE Analysis task is first composed of an anomalies analysis step based on results of the AS-IS Analysis report. Its aim is to produce a list of anomalies in the operations of functions of the organization and to help in the formulation of an amendment plan. These conclusions are submitted to managers who will come back with an approved plan for restructuring part or all of the existing system. The second component of the TO-BE Analysis task is a step called new system specification in order to produce the (functional, information, and dynamics) models of altered functions of the new system.

The overall architecture of the TO-BE Analysis task is shown in Fig. 18.

Organization Chart:



General Functional Diagram:

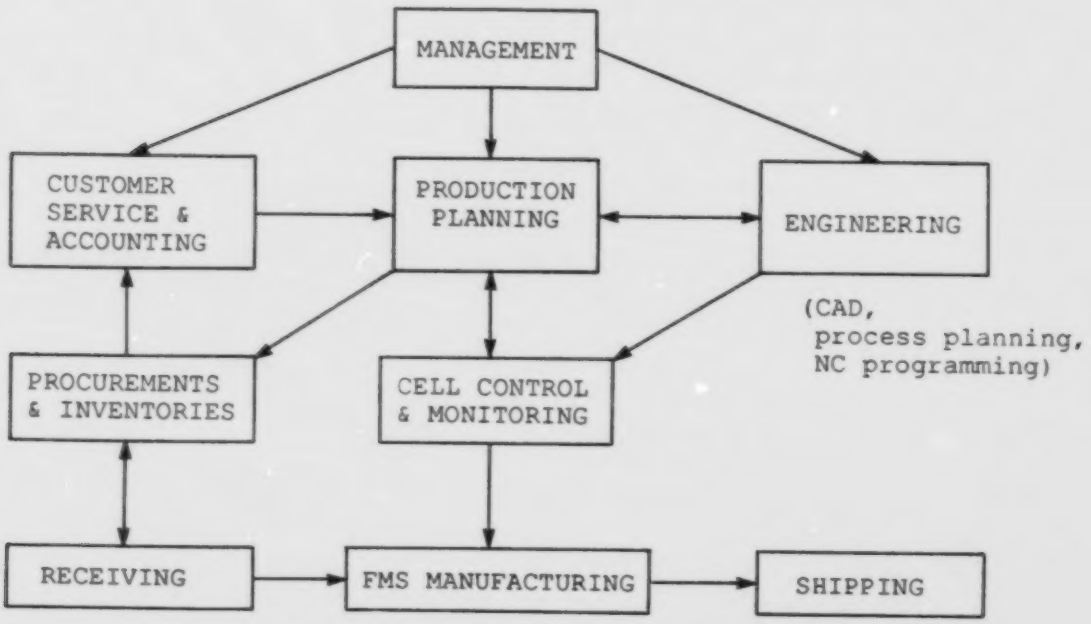


Figure 16: Top-Down Analysis of the ABC Company

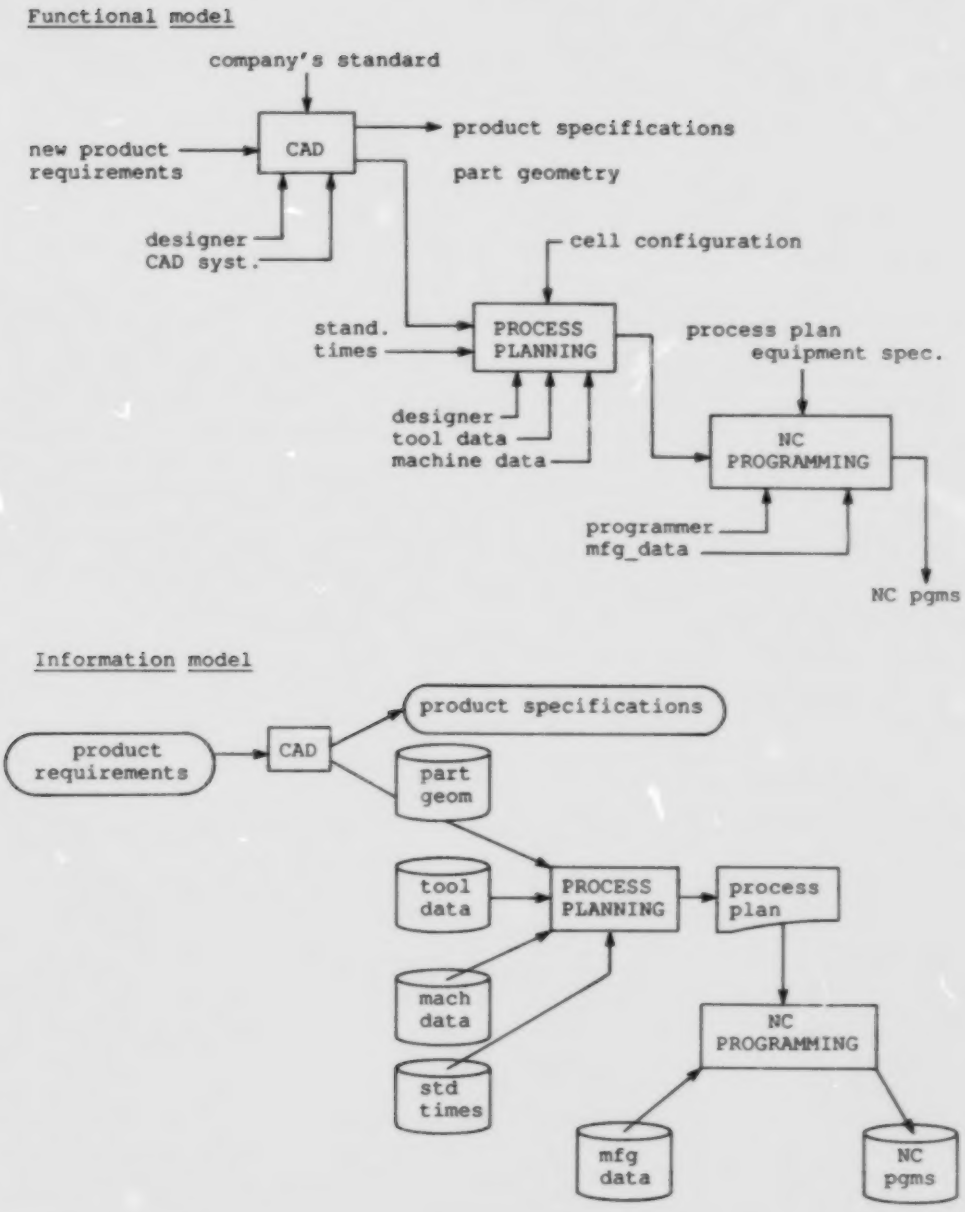


Figure 17: Bottom-Up Analysis of the Engineering Function

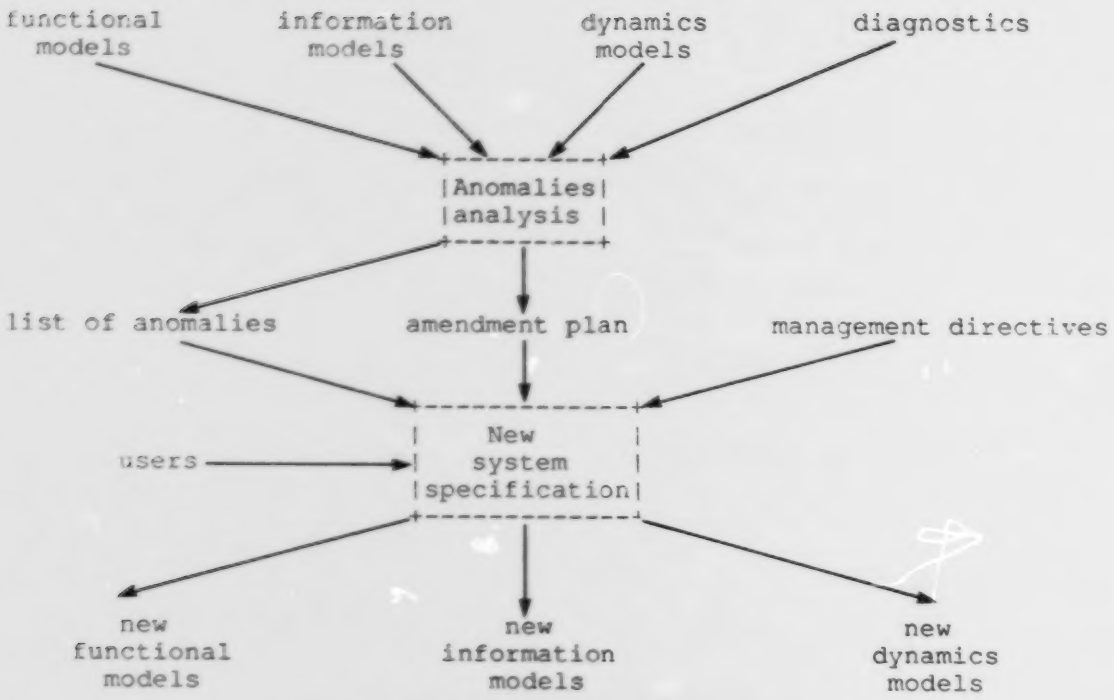


Figure 18: TO-BE Analysis Task Overall Architecture

7.3.1. Task 1.2: Input Description

The inputs of the TO-BE Analysis task consist of the outputs of the AS-IS Analysis for the anomalies analysis step plus managers' directives and users' advices for the new system specification step.

7.3.2. Task 1.2: Output Description

The output of the TO-BE Analysis task consists of a report, called the TO-BE Analysis report, which contains the new description of the enterprise in terms of:

- A list of anomalies or malfunctions
- An amendment plan suggesting how to correct the anomalies
- A set of new functional models, information models, and dynamics models which specify the new structure of functions which have to be modified in the context of the whole enterprise organization and according to automation priorities.

In some cases, the list of anomalies and the amendment plan may constitute an intermediate report. In other cases, the intermediate report and the TO-BE report may constitute only one document. This question is obviously dependent on the analysis context and the size of the enterprise which may vary from one analysis case to another.

7.3.3. Task 1.2: Description of Activities

The procedural description of the activities of the TO-BE Analysis task is shown in Fig. 19.

The goal of this phase is to specify the new system according to the results of the AS-IS Analysis phase. It involves two steps:

```
[anomalies analysis]
1. anomalies identification

for each function
   1.1 identify structural anomalies
   1.2 identify operational anomalies
   1.3 identify information processing anomalies

end for

2. propose an amendment plan

[new system specification]
3. function restructuring

for each function to be restructured
   3.1 define the new functional model
   3.2 define the new information model
   3.3 define the new dynamics model

 end for

4. validate models and check new system consistency
5. write the TO-BE Analysis report
```

Figure 19: TO-BE Analysis Activities

1. Identification of problems or anomalies of the current system: The models and the diagnosis elaborated during the AS-IS Analysis can be used to identify:
   - Structural anomalies (e.g., bad definition of a function)
   - Operational anomalies (e.g., non-productive operations)
   - Information-based anomalies (e.g., inadequate forms for data input)
2. Specification of the new system: A report must be written in order to specify the anomalies and how they will be corrected (amendment plan). Subsequently, a new system must be designed which will differ more or less from the existing one. On the basis of the AS-IS Analysis report and the amendment plan, top management will decide which functions will be modified or automated and which will not. It now becomes obvious that the AS-IS Analysis must be as accurate as possible, and must reflect the opinion of more than one person to efficiently support management decision-making. Also, the decision will be based on a global understanding of the complete operations of the company. This decision will be followed by the specifications of the new system which will be expressed in terms of the same tools as those used for the AS-IS Analysis. Then, the specifications will be transmitted to engineers for implementation. This is the reason why the tools have been designed to be useful to and be understood by managers, engineers, and technical people.

So far, we have discussed the tools in the case of enterprise improvement situations. Obviously, the tools can also be used for the design of new systems from scratch. Since no AS-IS Analysis results are available for this kind of situation, the specifications of the new system will be based on management decisions and the experience of the engineers.

7.4. Task 1.3: Environment Specification

When the new system has been defined on paper, one of the first steps for its implementation is the definition of the database environment. Thus, *a database committee* will be set up; then, the scope of application of the underlying database must be precisely specified in order to determine the limits of work of the

database committee; and finally, the environment database requirements will be collected. The first purpose of the Environment Specification task is to support such a transition activity, i.e., moving from the enterprise analysis usually involving managers' and system analysts' jargon to the conceptual database design usually involving computer scientists' jargon. The second purpose of the Environment Specification task is to realize the data requirements specification and collection phase as found in conventional database design methodologies.

The parts of the organization to be automated by means of the database application must be defined in terms of *organization environments*. An *environment* represents a set of highly-interrelated organizational functions (each considered as a black box) which must be executed to achieve an organization goal. A list of *users* and a list of *constraints* or function operations can be associated with each environment. A user is a person (or a set of people) who is actively involved in the operations of some environment. A constraint is a condition that must be satisfied by the environment functions. At this level, a constraint is usually expressed by means of a natural language sentence. An environment may not necessarily coincide with an enterprise department and it usually consists of several functions. But, an environment consists of at least one function.

Since users, operations, constraints, and objects have been well-identified for each function of the enterprise during the AS-IS and TO-BE Analyses, it is now easy to catalog information objects required by each organization environment and to identify data requirements. Therefore, the aim of the requirements collection planning step is to set up a collection plan which identifies requirements to be collected and users from whom requirements can be obtained. Finally, the collection step is the actual task of gathering these requirements which will be analyzed during the Conceptual Design phase.

The role of the Environment Specification task is thus to realize a logical link between Enterprise Analysis and Modelling and database Conceptual Design since it takes as inputs the documents issued from the AS-IS and TO-BE Analyses and it produces a description of the enterprise environments together with a set of requirements, in order to prepare the conceptual design phase.

The overall architecture of the Environment Specification task is displayed in Fig. 20.

7.4.1. Task 1.3: Input Description

The inputs to this task are the functional models, the information models, and the dynamics models, which describe the overall organization of the object enterprise, as produced by the TO-BE Analysis task.

7.4.2. Task 1.3: Output Description

The output of the Environment Specification task consists of (a) an organization network, (b) a set of environment specifications expressed by means of special forms specific to the M* methodology, and (c) a set of nonformatted and formatted requirements for each function of each environment. Examples of the forms used for (b) and (c) are given in the Appendix at the end of this report.

The organization network indicates the environments which will be concerned by the database application and the information objects exchanged by the environments. A complete description of the organization network formalism will be given in Section 7.4.4.

The forms provided by M* can be used to make an enterprise description for the database. The description will be composed of an environment list (giving a code number, the name, and a description of each environment) and an environment/functions cross-reference table (indicating which functions belong to which environment).

For each environment, the list of its functions and the list of its users are established and the function/users cross-reference can be produced. The list of forms to be collected in the environment is also established. For each environment, the description of each function is provided. The function description consists of nonformatted requirements (written or taped explanations in natural language of function procedures) and formatted requirements (examples of forms with explanation of their processing) collected within the environment. M* special forms also exist for requirements collection and classification.

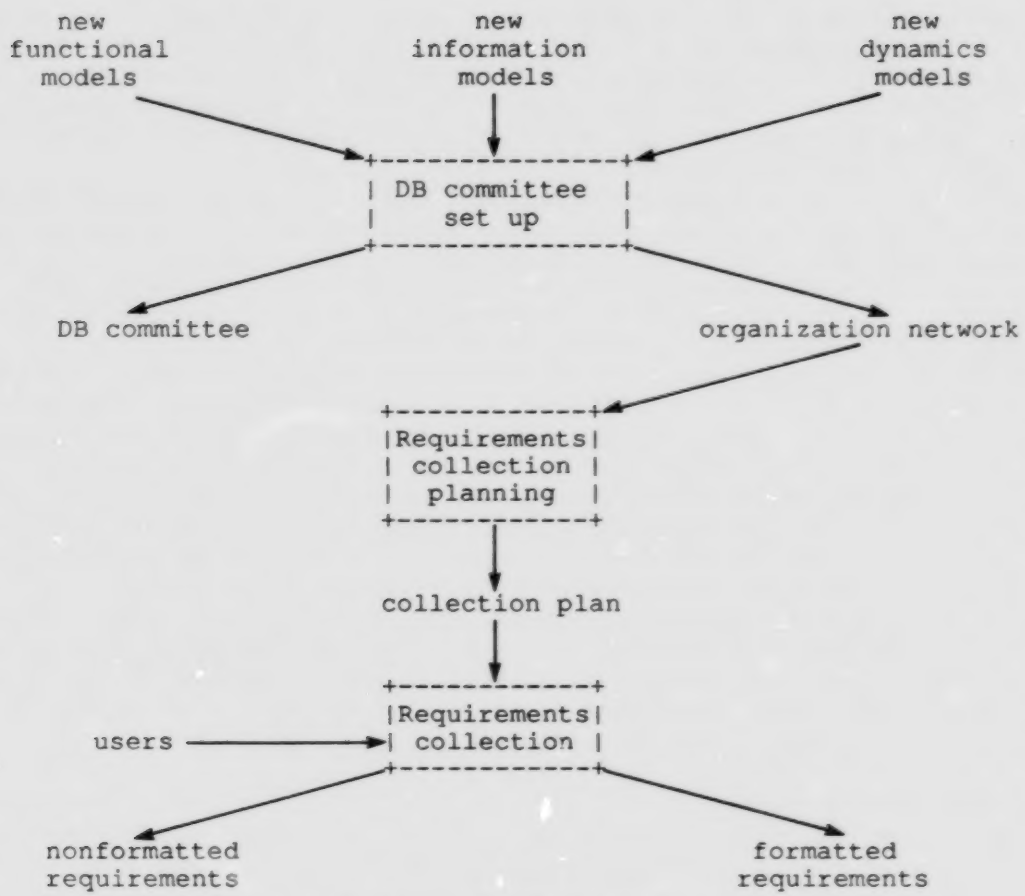


Figure 20: Environment Specification Overall Architecture

7.4.3. Task 1.3: Description of Activities

The procedural description of the activities of the Environment Specification task is shown in Fig. 21.

Each activity is described in more details in the following paragraphs.

7.4.4. DB Committee Set Up

In the DB committee set up activity, a database committee is first formed. The database committee will perform the rest of the Environment Specification task, but more importantly, it will performed the Conceptual Design of the database, the choice of the physical database system (DBMS), the Implementation Design, and plan the installation of the physical system.

Usually, this committee is composed of five to eight persons. One is a computer specialist and an expert on database technology. This person usually becomes the Database Administrator (DBA) and will chair the committee. The other persons assist the DBA in the design of the database. They do not need to be database experts but they must understand the technology and database design methodologies such as M* in particular. What is really important is that this group of people must be very representative of all departments, applications, and user functions affected by the future database system. Therefore, members of the committee will be managers, engineers, technicians, and support people from the company. The committee must not be too large to remain efficient, it must not be too small, but it must be very representative of any

```
[DB committee set up]
1. set up DB committee
2. establish the environment/functions cross-reference
3. produce the organization network

[requirements collection planning]
4. prepare the collection plan

for each environment
   4.1 establish the list of functions and users
   4.2 establish the list of forms to be collected
   4.3 plan requirements collection

end for

[requirements collection]
5. collection activity

for each environment and
 for each function

  5.1 interview users and collect relevant information
  5.2 collect the forms used in the function activities

 end for
end for
```

Figure 21: Environment Specification Activities

applications and/or users that will be concerned by the database system to ensure that their data requirements will be taken into account.

The database committee is not necessarily identical to the analysis team which did the Enterprise Modelling and Analysis. In fact, in practice it is never the case since the enterprise analysis is better performed if the analysts come from outside the enterprise while database design must involve people working in the day-to-day operations of the enterprise. However, if any continuity can be realized, this will strongly augment the chance for success of the project. This continuity can be assumed by the DBA who might have been involved in the enterprise analysis. Of course, the continuity is also assumed if both analyses are done by a private consulting company.

Once the database committee has been established, its first activity is to define the limits of the database application to be developed and to define the enterprise environments. This activity is executed on the basis of the results reported in the TO-BE Analysis report and from management directives about the part of the enterprise to be automated. Therefore, the list of environments is established and reported in the Environment List form of the M* methodology. An environment/functions cross-reference table indicating the name of each environment and the names of its related functions is also produced (see examples of the forms used in M* in the Appendix).

As an example, the environment/functions cross-reference table of the FMS example is shown in Fig. 22.

In order to complete this activity, an *Organization network* can also be defined to give a graphical representation of the environments, their relationships, and the flow of information objects circulating among them. However, for the sake of simplicity, the organization network defined at this level just describes the

ENTERPRISE DESCRIPTION
Environment/functions cross-reference

Environments:
S1: Process Planning
S2: Production Planning
S3: Cell Control and Monitoring
S4: Manufacturing

Functions:
F1: Process planning
F2: Process plan archival
F3: NC programming
F4: Production requirements analysis
F5: Workshop scheduling
F6: Cell control
F7: Cell management
F8: Machining
F9: Material handling

Env.Funct.	F1	F2	F3	F4	F5	F6	F7	F8	F9
S1	*	*	*						
S2				*	*				
S3						*	*		*
S4								*	*

Figure 22: The ABC Environment/Functions Cross-Reference

environments which will be affected by the database system to be defined, i.e., it only deals with the part of the enterprise which will be automated. Therefore, the organization network is the tool used to precisely define the scope of the future database application. Due to its expressive simplicity, this is the ideal tool for making the interface between management and database worlds or, in terms of the M* methodology, between Enterprise Modelling and Analysis (or organization level) and Conceptual Design (or conceptual level).

An Organization Network is a bipartite directed graph (E, O, A), where E and O are two disjoint sets of *environments* and *objects*, A is a set of *arcs* connecting an environment to an object or vice versa. Objects are the same elements as those used in the information model (presented in Section 5.2.2) and include messages, forms, and data. Graphically, an environment is represented as a rectangular box, and objects have the same representation as the one used in the information model. Therefore, the organization network of the enterprise directly derives from the TO-BE information models.

For instance, Fig. 23 shows the organization network for the part of the ABC company that we assume we want to automate. Manufacturing and Cell Control and Monitoring are environments which share the objects "commands", "materials", "resources", and "mfg_data". These objects form the *interface* between the two environments.

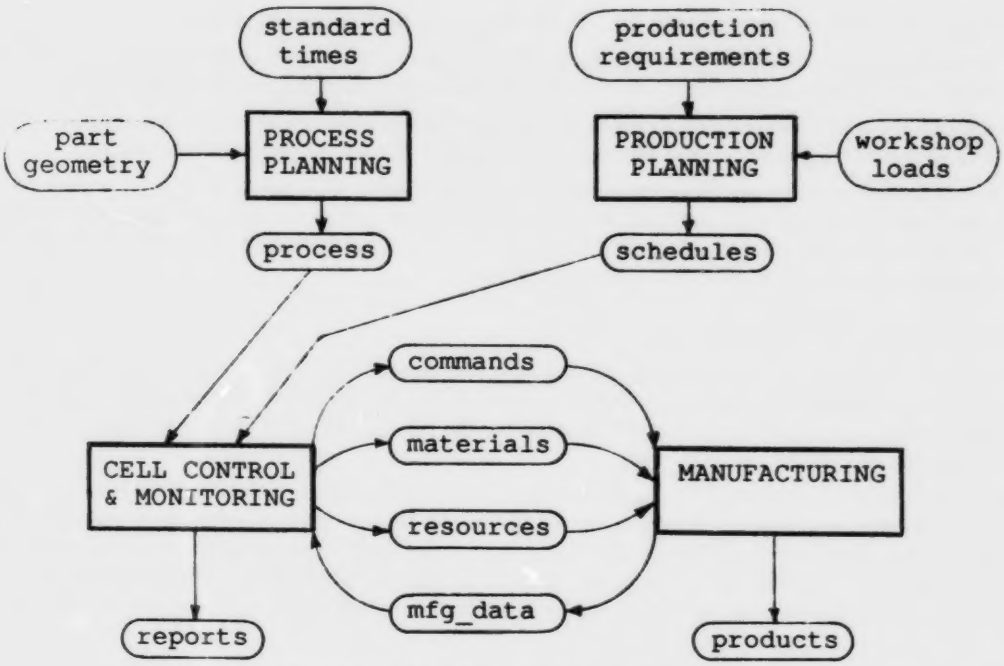


Figure 23: The ABC Organization Network

7.4.5. Requirements Collection Planning

In the M* methodology, two classes of requirements to be collected are considered:

1. Formatted requirements: They refer to structured documents that can be described by means of a limited number of syntactic rules. They can be classified as (although the following list is not at all exhaustive):
   - Forms for data acquisition and output description (e.g., customer order forms, delivery slips, etc.)
   - Record formats from data files (e.g., payroll file, bills of materials files, etc.)
   - Data Division from COBOL programs or other similar languages
   - Data schemata defined with data description languages of pre-existing databases
   -Procedures, expressed in terms of data manipulation languages or query languages
2. Nonformatted requirements: They result from interviews, questionnaires, and existing written descriptions. Such types of requirements are usually expressed by means of natural language sentences and are called *user requirements* in what follows.

The requirements collection planning activity is concerned with the preparation of a collection plan established from the specification of the organization network and from results of the AS-IS and TO-BE Analyses. For each environment, the collection plan identifies the requirements to be collected and their class, and the users from whom detailed data and processing requirements about the functions that must be automated can be obtained.

To support the collection planning activity, M* provides a series of forms including:

- a functions/users list which gives a code number, a name, a short description, and the list of users for each function and which gives a code number, a name, and a short description for each user
- a forms list which indicates all forms to be collected for a given environment. Each form is listed with a code number, its name, a short description, and the list of users working with this form
- a requirements collection form for nonformatted requirements of a function of a given environment. The form can be prepared with indication of the environment code, the function code, the user code of the user to interview (or the document to read). The requirements description is left blank and will be completed during the requirements collection activity

7.4.6. Requirements Collection

This is the actual task of requirements collection. It requires one to interview users, read user reference manuals for procedural operations, etc. During interviews with users, a more detailed analysis of their work or their operation procedures than the one done during the AS-IS Analysis is performed, but it is done from an information processing viewpoint only.

Figure 24 gives an example of a formatted document, in this case a process plan form as used in the process planning function, and Fig. 25 illustrates the nonformatted data requirements for the production function. Figure 25 also gives an illustration of the structure of forms used in M* to collect nonformatted requirements. The form indicates the environment name, the function name, and a textual description of the requirements as obtained from user interviews or from analysis of enterprise procedure documentation.

```
Version No.:                                Date:
Designer name:


Part_id: FICBLOC                    Product_design #: 125

Cell_id: 5                          Process_plan #: 27


Oper.#  Oper.name          Machine_code          NC_progr.code

  1     turn diameter       1440-lathe            FICBLOC_L1
  2     drill hole          W50-mach.centre       FICBLOC_M1
  3     measuring           8202-B&S              FICBLOC_CMM1
```

Figure 24: The Process Plan Form

7.5. Conclusion

In this Section, we have discussed the series of tasks involved in Enterprise Modelling and Analysis of Production Systems with respect to the methodological framework summarized in Section 3.3. The AS-IS Analysis task provides decision-makers with a "snapshot" of the current status of the functional structure of the enterprise. The role of the TO-BE Analysis task is to support the decision of what has to be automated with the help of database technology and how it must be automated. Finally, the Environment Specification task translates management decision and analysis into database-oriented requirements and prepares the Conceptual Design phase of the enterprise database.

REQUIREMENTS COLLECTION Environment: Manufacturing

Function: Production User: ...

A fabrication lot is a batch of similar parts to be produced. Each occurrence of a part of the same lot is a workpiece made of some material. The workpieces are manufactured according to a predefined sequence of machining operations specified by a process plan. A workpiece must be mounted on the appropriate device to be processed on a given machine with a given tool. After each operation, the workpiece is prepared for the following operation until the process plan is completed.

Figure 25: Nonformatted Requirements Collection Example

The major originality and strength of the M* methodology is to recognize the importance of performing a preliminary detailed enterprise analysis study before starting the database design in order to clearly identify the exact requirements of the database system with respect to the complete enterprise objectives. The database design phase then logically follows the enterprise analysis phase. Therefore, M* really realizes the integration of Production System Analysis with Conceptual Database Design as required by CIM database environment development.

8. Phase 2: CONCEPTUAL DESIGN

8.1. Introduction

In the Conceptual Design phase a set of *local views* (one for each environment) and a *global view* of the object enterprise are constructed. A *view* (both local or global) is the complete and formalized representation of the static (data) and dynamic (operational and functional) environment requirements.The description of a view contains: (a) a *data schema*, (b) a set of *operation schemata*, and (c) a *functional net*. The concept of functional net is introduced in Section 8.3.

First of all, formatted and nonformatted requirements collected in the previous phase are analyzed during the Form Analysis task and the Requirements Analysis task. Static and dynamic requirements are expressed in terms of the data model, operation model, and process model of the M* integrated model (see Section 5.3). In order to support the Form Analysis, a *form model* will be introduced.

After the analysis of forms and user requirements, the View Design task produces a model of each environment of the enterprise consisting of a *local view* expressed in terms of the EER model (see Section 5.3.1).

The View Integration task constructs the *global view* by aggregating all the local views.

The overall architecture of the Conceptual Design phase is shown in Fig. 26.

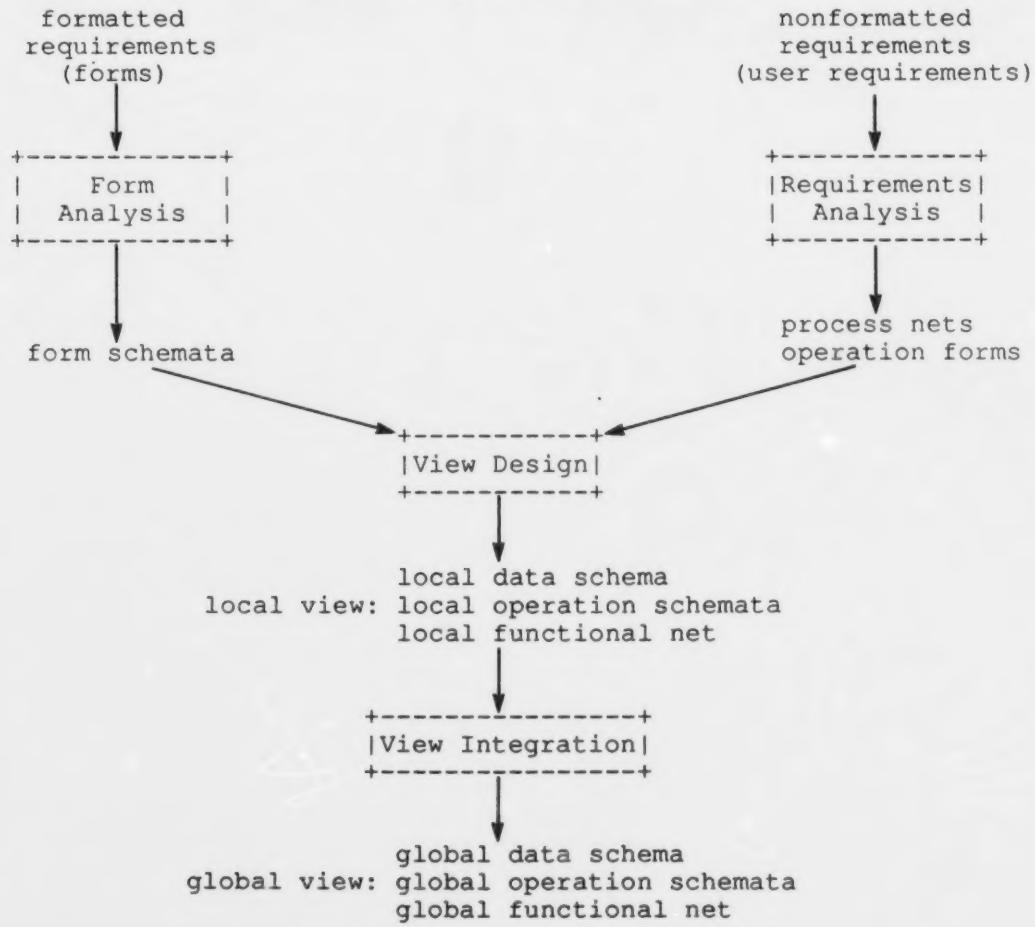


Figure 26: Conceptual Design Phase Overall Architecture

8.2. Task 2.1: Form Analysis

In this task, we consider all kinds of *forms* or documents which are used in the day-to-day operations for data collection and communication, and typically found in office environments.

Starting from the forms collected from the enterprise, a set of data schemata (one for each form) is produced.

A complete methodology to transform form descriptions into data schemata expressed in terms of the EER model has been illustrated in ref. 43. We will summarize here the basic steps of this methodology for the sake of completeness. Furthermore, these steps will be adapted to the M* global design environment.

The Form Analysis task consists of three activities: (a) *form analysis*, in which a form is segmented into meaningful parts, called areas; (b) *area design*, in which the data structures contained in each area are represented by an area schema; and (c) *aggregation and restructuring*, in which area schemata are merged and restructured into a form schema. The overall architecture is displayed in Fig. 27.

8.2.1. Task 2.1: Input Description. The Form Model

Input to the Form Analysis task consists of the sets of forms collected from each environment. Forms are formatted documents that are used within the organization to manage communications (in input, in output, and between functions). Since data are well-structured in forms, the design process is easy and well-defined compared to the design process for nonformatted requirements.

Operational requirements on forms (usually expressed by means of natural language descriptions) are considered in the Requirements Analysis task and integrated with form schemata in the View Design task.

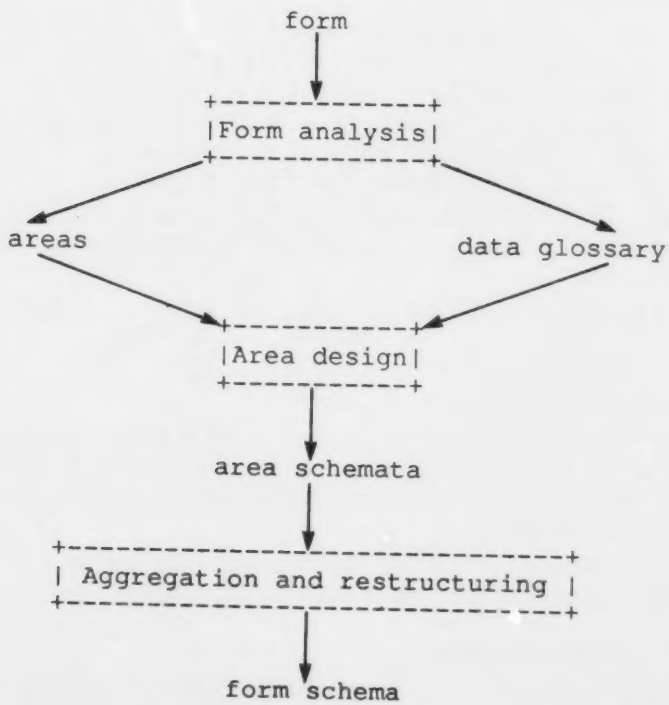


Figure 27: Form Analysis Overall Architecture

The *form model* assumes that any form can usually be divided into data *areas*. A data area is a section of the form which contains information pertaining to the same argument or describing the same concept. Areas may be further structured into subareas to any level of subdivision.

Data areas can belong to one of the four following types:

1. *Certificative parts*: They contain information elements like signatures, dates, marks, and so on, which are usually not relevant to the data design process.
2. *Extensional parts*: They concern the sections of the form that must be filled in by the user when the form is completed.
3. *Intensional parts*: They contain implicit or explicit references to symbols identifying extensional parts.
4. *Descriptive parts*: They specify instructions or rules to be followed when filling in the extensional parts.

Problems with forms come from the great variety of different configurations they can present. Furthermore, semantic information concerning the structure and the meaning of a form can be explicity expressed or can be contained in common implicit conventions.

Usually extensional and intensional parts are mixed and interconnected in a form. A classification of information that appears in extensional/intensional parts of a form depends upon the type of linguistic representation used:

a. Parametric text: It is a text written in natural language with fields that are to be filled with values extracted from suitable domains (e.g.; We certify that M. ..... was born in ....). Once such fields have been completed, the text appears as a set of coherent sentences in natural language. Frequently, in such cases, the intensional information (name, city, date_of_birth,...) is not explicitly present and must be elucidated during the design process.

b. Structured frame: It is the bidimensional representation of n-dimensional tables. In the case of n > 2, a table is usually expressed by means of ordered sets of bidimensional tables.

Descriptive parts usually concern rules and/or constraints on the way forms must be used and filled in with data. They can be interconnected with extensional/intensional parts or they can be mentioned apart (e.g., footnotes). Frequently, rules are not indicated at all since they are based on "obvious" conventions (e.g., do not complete the field: "Number_of_pregnancies", if the person is a male).

Usually rules are translated into constraints which can be expressed in the form schema or must be enforced within application programs. As discussed in Section 5.3.1, a relevant set of constraints can be directly expressed in the EER model.

Concepts introduced by the form model will be illustrated hereafter with examples selected from the case study.

8.2.2. Task 2.1: Output Description

The output of this task consists of a set of form schemata for each environment. Form schemata are expressed by means of the EER model. A form schema represents a formal structure of the form organization in terms of information objects (entities, relationships, attributes, ...). An example of a Form Schema form can be found in the Appendix.

A data glossary is also produced for each form. The structure of the forms used in M* for form data glossaries can also be found in the Appendix. Use of these documents is explained in the following Section.

8.2.3. Task 2.1: Description of Activities

A procedural description of the activities of the Form Analysis task is shown in Fig. 28.

During the Form Analysis activity, forms are analyzed to determine the form areas and to classify the different areas according to the four types discussed previously. Areas may be further broken down into

```
for each form:

 [form analysis]
  1. distinguish parts (descriptive, extensional/intensional)
  2. select areas and subareas, and give them names
  3. for each area
       3.1 extract elementary concepts and their properties
       3.2 fill in the data glossary
     end for

 [area design]
  4. for each area
       repeat 4.1 choose a type for a concept in the glossary
              4.2 insert the concept in the area schema
              4.3 update the glossary
       until  4.4 all the concepts have been inserted
     end for

 [aggregation and restructuring]
  5. merge area schemata
  6. analyze form schema
       6.1 enrich form schema
       6.2 restructure form schema

end for
```

Figure 28: Form Analysis Activities

subareas. For each area or subarea, an area name must be provided which expresses, in the most explicit way, the meaning of the related concept.

During area analysis, the *form data glossary* must be filled in. The glossary describes the elementary concepts (i.e., entities, relationships, attributes, and abstraction hierarchies) contained in the form and their properties.

The data glossary is used in the area design activity for the construction of area schemata which are then aggregated and (possibly) restructured (during the aggregation and restructuring activity) to produce the final form schema.

As an example, let us assume that we want to process the form of Fig. 24, concerning a simple Process Plan produced by the Process Planning environment.

During steps 1 and 2 of Fig. 28, forms are analyzed in detail in order to describe their structure in terms of the form model, i.e., part nature, areas, and elementary concepts. In Fig. 29, areas and subareas are enclosed within dashed lines.

Step 3 depends on the type of linguistic representation (parametric text or structured frame) contained in the area. For parametric text, usually blank fields must be filled in with atomic values. The names of fields (identifying the elementary concepts) appear implicitly or explicitly within the text. For structured frames, users have to fill in n-dimensional tables which are described by means of table and column names and indexed by means of (limited) value sets. The meaning of the indexing sets and the names usually suggest the elementary concepts that are dealt with.

Elementary concepts must then be inserted in the form data glossary. Each entry contains a code, the concept name, a brief description, a domain description, possible instances and synonyms, and the abstraction

hierarchies in which the concept is involved. Description, synonyms, and hierarchies are used for a better understanding of the concept meaning.

In the example, the data glossary for the Process Plan form is shown in Fig. 30. The structure of this document follows the structure of the Data Glossary form presented in the Appendix.

In step 4, a draft area schema is constructed. Each elementary concept must be classified (as an entity, an attribute, a relationship, or a hierarchy) and inserted into the draft schema, using the glossary as a source of information and cross-reference. For this reason, it is important to keep the glossary up-to-date (step 4.3).

Step 4.1 is complex and difficult to formalize. The classification process, which assigns a type to each concept, can be driven by the *conceptual trees* that represent the structure of the areas. For the Process Plan form of Fig. 29, the conceptual trees are shown in Fig. 31.

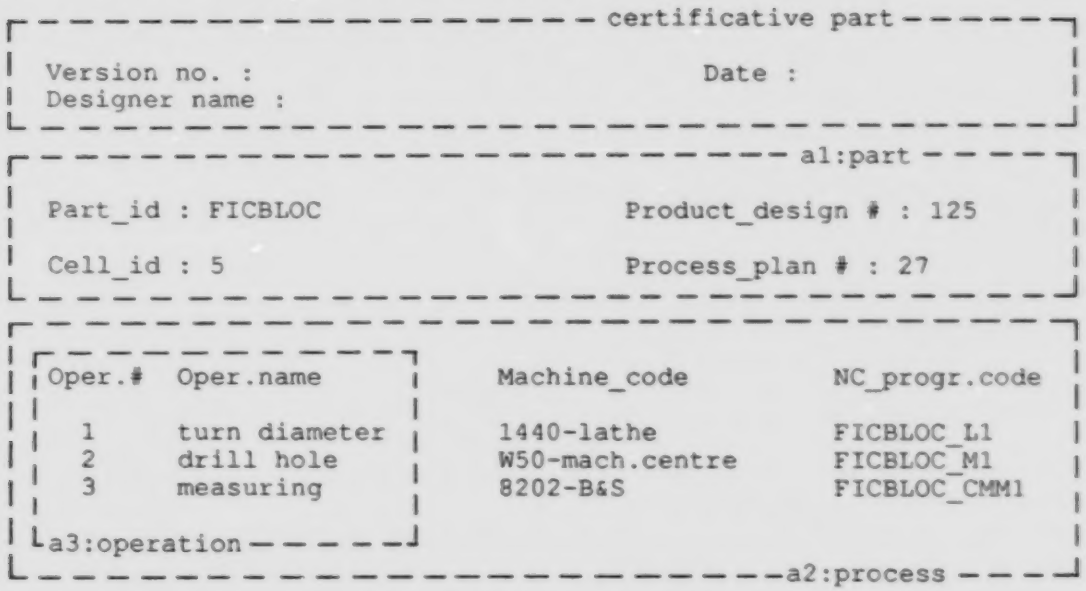


Figure 29: The Process Plan Form

DATA GLOSSARY — Environment : PROCESS_PLANNING; Form : Process Plan

code	name	description	domain	instance	syn.	hierar.	ref.
D1	part_id	part_identific.	C8	FICBLOC			a1
D2	cell_id	cell_identific.	int 2	5			a1
D3	prc_pl#	process_plan id.	int 5	27			a1
D4	wrkp#	worpiece id.	int 2	2			a1
D5	prd_ds#	product des. id.	int 4	125			a1
D6	oper#	operation id.	int 2	1,2,3			a3
D7	oper_nm	operation name	C20	measuring			a3
D8	mach_cd	machine code	C20	8202-B&S			a2
D9	NC_prcd	NC program	C15	FICBLOC_L1			a2

Figure 30: The Process Plan Data Glossary

Usually, concepts at the lower level of a tree refer to attributes of entities which are represented as concepts at some internal levels of the tree.

A concept can be classified as an entity if it has properties and if it does not seem natural to define instances for it. It can be classified as an attribute if it has no properties, it is natural to find another concept for which the attribute is a property, and it is easy to give sample values of it (with their corresponding domains) from the examples available. For instance, in Fig. 31, the concept "cell_id" is an attribute (an instance may be the two-digit integer 2), and we have to introduce the entity it belongs to (which has been called CELL) in the area schema.

During step 5, area schemata are merged. Merging can be carried out by identifying the common concepts that appear in the two merging schemata. Common concepts usually have the same name in different area schemata of the same form. However, concept types can be different (i.e., a concept is represented as a compound attribute in the first schema, and as an entity in the second). In this case, a transformation must be applied to unify the representations. Merging can then take place by superimposing common concepts.

In Fig. 32, the Process Plan form schema is illustrated.

The last activity concerns the enrichment and the restructuring of the form schema.

The first step (enrichment) is devoted to schema completion, that is, the specification of all the properties of attributes, entities, relationships, and hierarchies that can be present in an area schema and hidden from another. For instance, in this step all relationship cardinalities and optional properties of attributes are fixed.

The aim of the second step (restructuring) is to increase the clarity and the expressiveness of the schema by means of data modifications. Since clarity and expressiveness are difficult to quantify, this step is highly informal. Usually, a global analysis of the whole schema may suggest some restructurings. For instance, if two entities have many properties in common (while representing different concepts), they can be merged into a unique hierarchy. The introduction of new abstractions is justified when they add further clarity and simplicity to the schema. It is discouraged when "unnatural" entities are to be created.

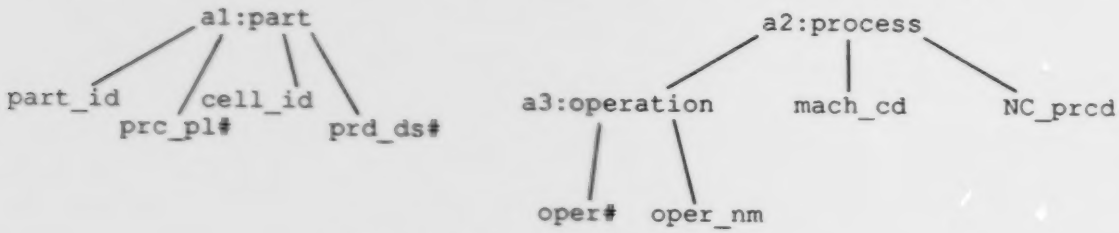


Figure 31: Process Plan Conceptual Trees

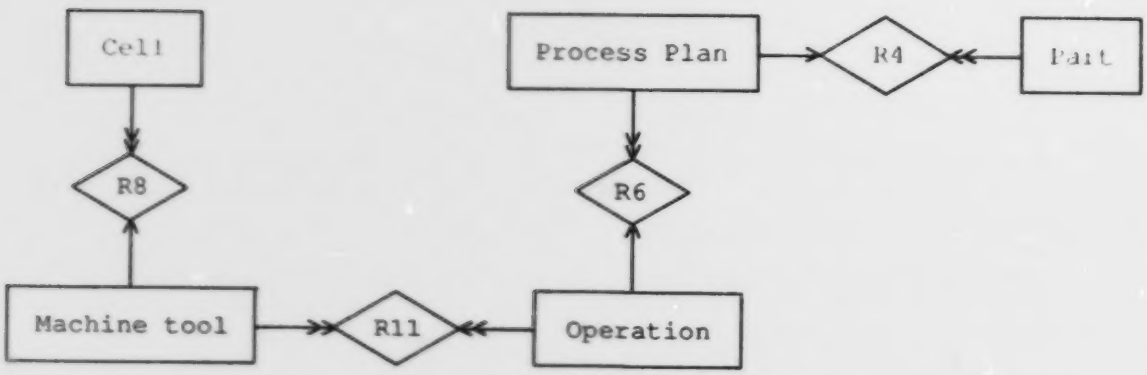


Figure 32: Process Plan Form Schema

8.3. Task 2.2: Requirements Analysis

The aim of the Requirements Analysis task is to extract as many data structural properties as possible from the description of the functional and operational characteristics concerning those parts of the enterprise which are to be automated by means of a database application.

User requirements have been collected, following a collection plan, by means of interviews, questionnaires, or already available documents (26). Users have been encouraged to describe how the organization functions are executed and to specify the ways in which data are captured, produced, or processed.

As noted in ref. 26, natural language descriptions must be restricted according to suitable conventions. The restriction process (or filtering activity) aims at improving comprehensibility and removing semantic ambiguity of the texts, for instance by:

- reducing repetitions, redundancies, and use of synonyms
- resolving homonyms by introducing suitable substitutes
- substituting pronouns with the corresponding nouns
- making explicit elements being implicit
- applying restrictions to articles, quantifiers, plurals, and so on

The filtered sentences must be classified either as *data sentences* or *operation sentences* and used to fill in *data* and *operation glossaries*.

For each operation, the *schema design* activity produces an operation schema describing the data required by the operation.

For each function, the *process design* activity generates a process net describing the function behaviour.

The overall architecture of the Requirements Analysis task is displayed in Fig. 33.

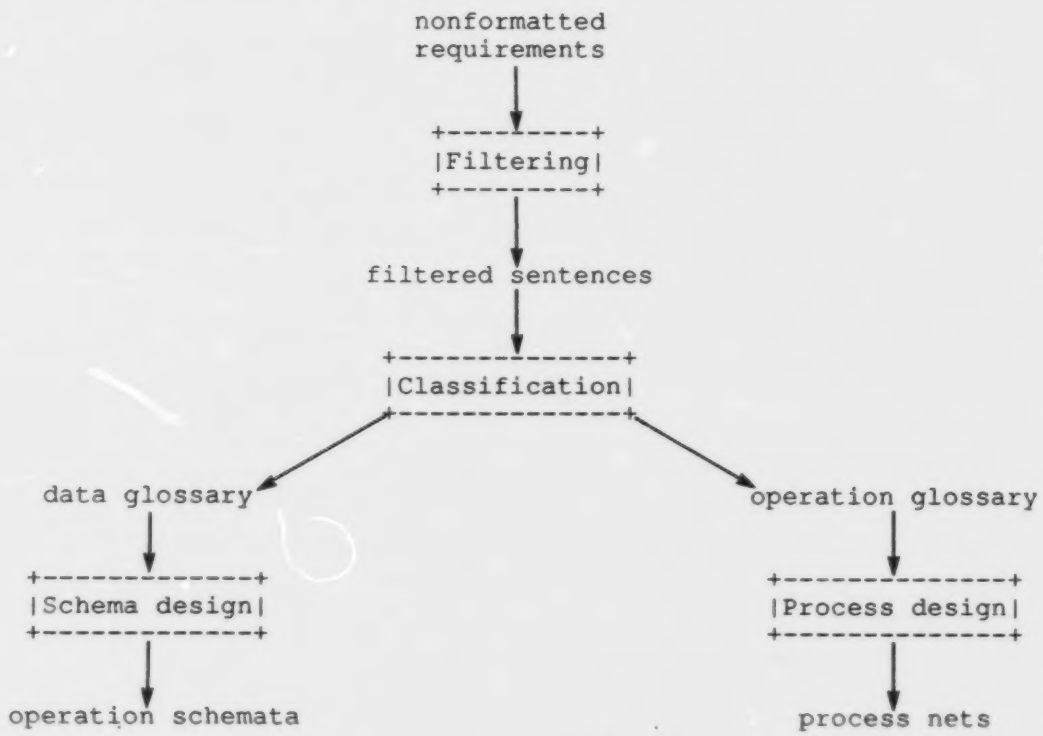


Figure 33: Requirements Analysis Overall Architecture

8.3.1. Task 2.2: Input Description

Input to this task is a set of natural language sentences, called nonformatted requirements. They have been collected during the Environment Specification task of the Enterprise Modelling and Analysis phase (see Section 7.4).

8.3.2. Task 2.2: Output Description

The output of the Requirements Analysis task is composed of a set of operation schemata and a set of process nets. In this way, dynamic properties of the object organization are described in order to identify the structural properties of data managed in the organization environments.

8.3.3. Task 2.2: Description of Activities

The procedural description of the activities of the Requirements Analysis task is shown in Fig. 34.

Let us consider the Manufacturing environment example. The sentences extracted from the description of the Production function (Fig. 35) have been classified and used to fill in the data and operation glossaries (Fig. 36).

As can be noticed, the sentence t4 has been added to the requirements to specify the meaning of the word "prepared". In fact, the requirement collection is usually an iterative process which "refines" the description of the user needs until no more interpretation problems exist.

In the verifying step of the classification activity, glossaries are checked for completeness and consistency, i.e., all data referred to in the operation glossary must be described in the data glossary.

The schema design activity constructs the operation schema for each operation described in the operation glossary. First of all, for each data item referred to in the operation description, a type must be selected (entity, attribute, or hierarchy). Entities are then logically connected by means of relationships.

```
for each environment and
 for each function

 [filtering]
  1. filter sentences and fill in sentence forms

 [classification]
  2. classify sentences
  3. fill in data and operation glossaries
  4. verify glossaries

 [schema design]
  5. for each operation
       5.1 assign a type to each datum involved
       5.2 construct the operation schema
     end for

 [process design]
  6. for each operation
       6.1 determine input and output conditions
           (messages and data)
       6.2 determine the rules that data must satisfy
     end for
  7. coordinate operations

 end for
end for
```

Figure 34: Requirements Analysis Activities

DATA REQUIREMENTS — Environment : Manufacturing

Function : Production — User : ...

code	sentence
d1	A fabrication lot is a batch of similar parts to be produced.
d2	Each occurrence of a part of the same lot is a workpiece made of some material.
d3	There is a predefined sequence of machining operations specified by a process plan.

OPERATION REQUIREMENTS — Environment : Manufacturing

Function : Production — User : ...

code	sentence
t1	The workpieces are manufactured according to a predefined sequence of machining operations.
t2	A workpiece must be mounted on the appropriate device to be processed on a given machine with a given tool.
t3	After each operation, the workpiece is prepared for the following operation until the process plan is completed.
t4	The preparation of a workpiece implies removing it from the device and checking if the necessary devices, tools, and machines are available for the subsequent operation of the given cell.

Figure 35: Requirements Collection

In the example, the operation schema corresponding to the operation T4 (Check) is shown in Fig. 37.

For each operation, the process design activity determines:

1. The messages and the data that are necessary to allow the operation to be activated (input specifications)
2. The rules that input data must satisfy
3. What changes are generated on the data (output specifications)

Operations are then coordinated by inspecting input and output specifications.

The process net corresponding to the Production function is illustrated in Fig. 38.

8.4. Task 2.3: View Design

In the View Design task, a local view is built for each environment of the enterprise. The local view is the formal representation of both the statical (data) and dynamical (operations and processes) requirements (36).

Starting from the operations schemata (constructed in the Requirements Analysis task) and from the form schemata (produced in the Form Analysis task), a *local data schema* or environment schema is generated.

DATA GLOSSARY — Environment : Manufacturing; Function : Production

code	name	description	domain	instances	syn.	subset	ref.
D1	FB_LOT	fabrication lot					d1
D2	PART	part					d1
D3	WORK_P	workpiece					d2
D4	mat	material	C(15)	steel			d2
D5	OPERAT	mach. operation					d3
D6	PR_PLAN	process plan					d3
D7	DEVICE	device					t2
D8	MACHINE	machine					t2
D9	TOOL	mach. tool					t2
D10	CELL	manuf. cell					t4

OPERATION GLOSSARY — Environment : Manufacturing; Function : Production

code	name	description	data involved	freq.	ref.
T1	Manuf	manufacturing process	D3,D5		t1
T2	Mount	mounting a workpiece	D3,D5,D7,D8,D9		t2
T3	Dismt	dismounting a workp.	D3,D5		t4
T4	Check	check	D3,D5,D7,D8,D9,D10		t4

Figure 36: Data and Operation Glossaries

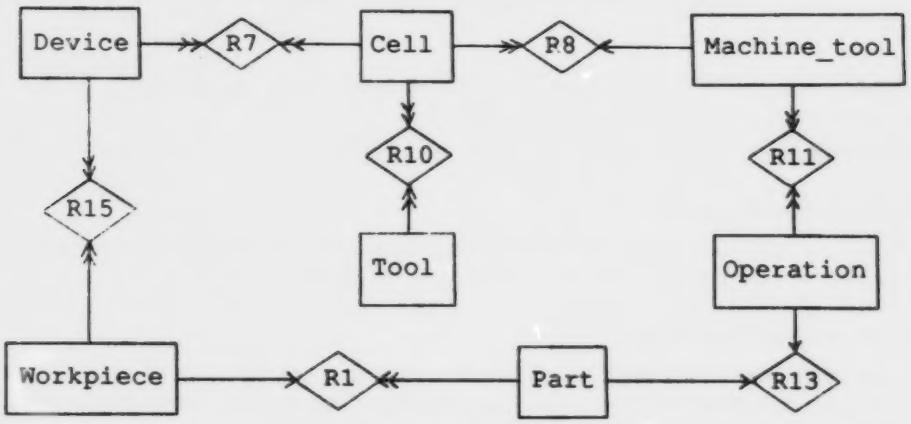
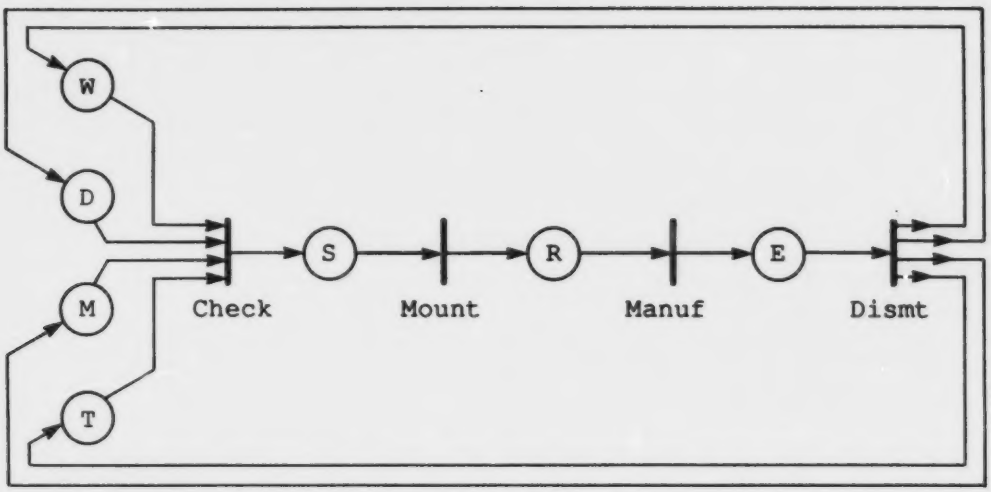


Figure 37: The Check Operation Schema

Moreover, in order to represent communications between the environment functions, the process nets are coordinated into the *local functional net*.

First of all, form schemata are aggregated to produce a draft local data schema. The aggregation process is then iterated for all the operation schemata.



Conditions:

- W: a workpiece is ready for machining.
- D: a device is ready to receive a workpiece.
- M: a machine is free.
- T: a tool is free.
- S: the workpiece can be machined and the proper device, machine, and tool have been reserved.
- R: the workpiece is mounted on the proper device.
- E: a workpiece has been machined.

Figure 38: The Process Net for the Production Function

The aim of the coordination of process nets activity is to analyze functions in the context of their environment and to point out the communication between them.

The overall architecture of the View Design task is shown in Fig. 39.

8.4.1. Task 2.3: Input Description

The input to this task consists of operation schemata and form schemata (expressed by means of the EER model) and process nets (expressed by means of the process net model).

8.4.2. Task 2.3: Output Description

For each environment, the View Design task produces a local data schema, a set of local operation schemata, and a local functional net. To describe the data schema and the operation schemata, the EER model is used. As regards the functional net, the process net model is used.

8.4.3. Task 2.3: Description of Activities

The procedural description of the View Design activities is shown in Fig. 40.

The aggregation of form schemata is the first activity because data appearing in forms are usually highly structured. Their conceptual analysis is then easier, and the draft schema produced is more reliable with respect to the schemata produced by the analysis of operations (described in a nonformatted way).

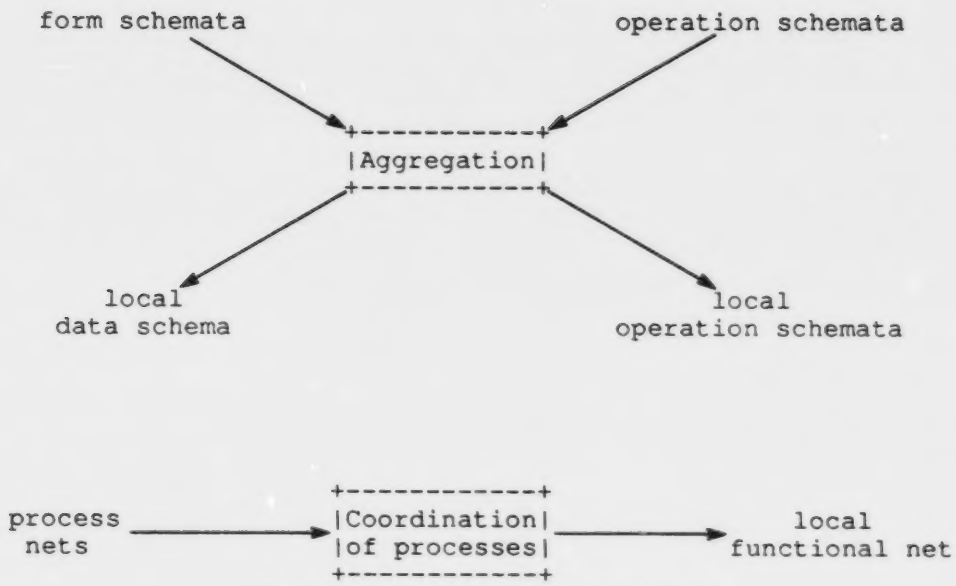


Figure 39: View Design Overall Architecture

```
for each environment:

[aggregation]
 1. collect form and operation schemata of the environment
 2. choose a form schema as the draft data schema
    repeat 2.1 choose a form schema
           2.2 find common concepts between the object schema
               and the draft data schema and solve conflicts
           2.3 merge the two schemata
    until  2.4 all form schemata have been processed
 3. repeat 3.1 choose an operation schema
           3.2 find common concepts between the object schema
               and the draft data schema and solve conflicts
           3.3 merge the two schemata
    until  3.4 all operation schemata have been processed
 4. restructure the data schema
 5. restructure the operation schemata

[coordination]
 6. coordinate process nets
    6.1 choose a process net as the draft functional net
    repeat 6.1 choose a process net
           6.2 find external conditions in common with the
               draft functional net
           6.3 coordinate the nets
    until  6.4 all the process nets have been processed

end for
```

Figure 40: View Design Activities

The aggregation process is carried out by identifying common concepts between the draft schema and the schema to be aggregated. *Conflicts* may arise because different forms or different operations may have different "views" of the same concept. Usually, it happens that the concept has different representation structures in the two schemata, so that the schemata must be updated to unify the representations before merging is applied.

The data schema is then analyzed. Restructuring may be applied to obtain a final schema which is clear and easy to understand. Finally, the operation schemata are restructured according to the changes on the data schemata performed during aggregation.

The data schema for the Process Planning environment is shown in Fig. 41.

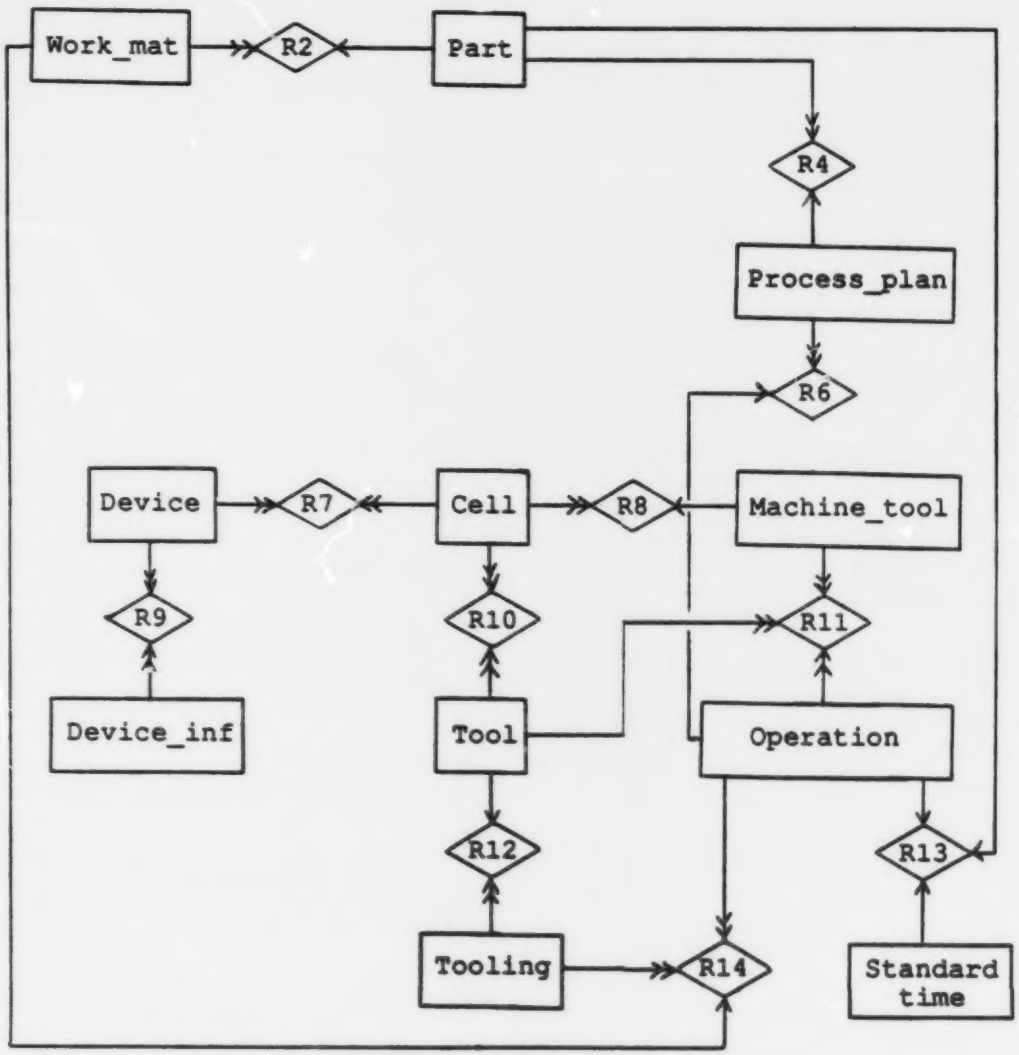


Figure 41: The Process Planning Environment Data Schema

In the activity 6, process nets are coordinated into the environment *functional net*. Process nets are analyzed in the context of their environment to determine communications requirements (both data and messages) between them. We have essentially two communication protocols (represented in Fig. 42):

1. *Synchronous communication*: The *sender* waits for an answer from the *receiver*
2. *Asynchronous communication*: The *sender* acts without taking into account the *receiver*

The functional net for the Process Planning environment is shown in Fig. 43.

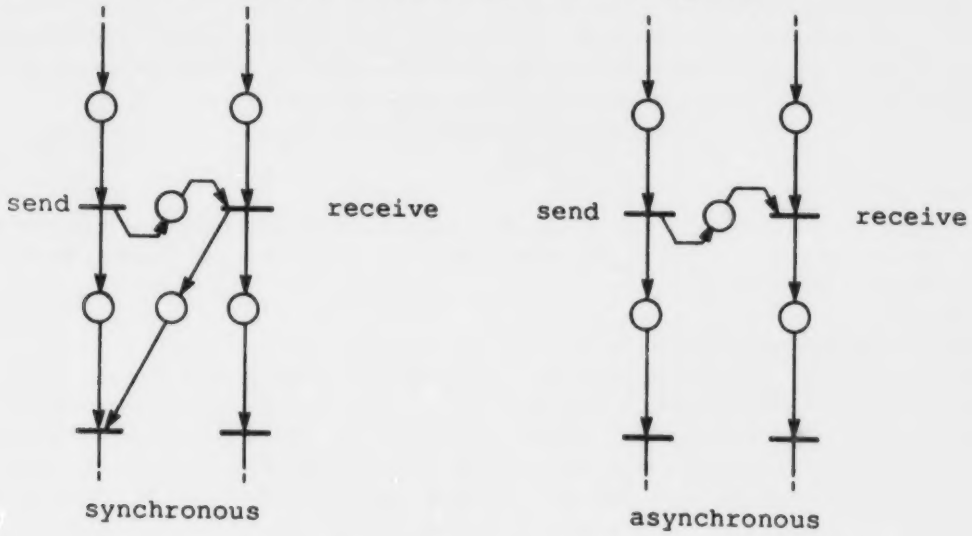


Figure 42: Communication Protocols

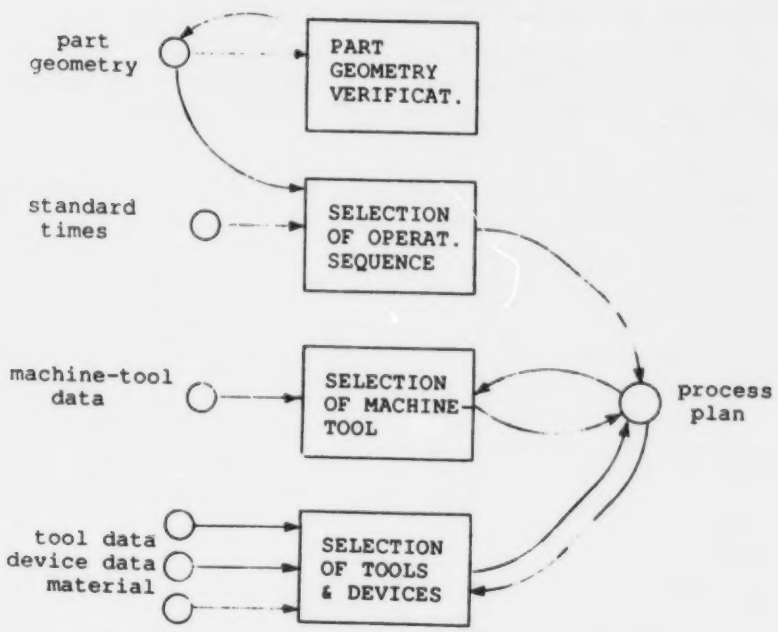


Figure 43: The Process Planning Environment Functional Net

8.5. Task 2.4: View Integration

The View Integration task integrates the descriptions of all the environments of the object enterprise. As usual, both static descriptions (local data schemata) and dynamic descriptions (local functional nets) are integrated to produce the global conceptual representation (global view).

First of all, the integration of local data schemata is carried out. After the global data schema has been produced, operations schemata must be restructured to make them coherent with the representation of data chosen in the global data schema.

The coordination of functional nets analyzes the communications (both data and messages) between environments. During this analysis, each environment is represented as a "macro" function because its internal structure (as a net of operations and conditions) is irrelevant at the global level.

The overall architecture of the View Integration task is illustrated in Fig. 44.

8.5.1. Task 2.4: Input Description

Input to this task consists of the local data schemata, the operation schemata, and the local functional nets. Data and operation schemata are described by means of the EER model, functional nets are described by means of the process net model.

8.5.2. Task 2.4: Output Description

The View Integration task produces the global data schema and the global operation schemata (which are described by means of the EER model) and the global functional net. As regards the global functional net, it is a high level net which describes the communications among the different environments of the object enterprise. Since at this level environment details are not interesting, the global functional net is a net in which environments are represented as "high level functions" in order to show just their communication behaviour.

8.5.3. Task 2.4: Description of Activities

The procedural description of the View Integration activities is shown in Fig. 45.

The first activity of this task is the integration of the local environment data schemata. This requires more care than the intra-environment aggregation carried out in the View Design task because users belonging to different environments may have different perceptions of the same concepts, giving rise to different

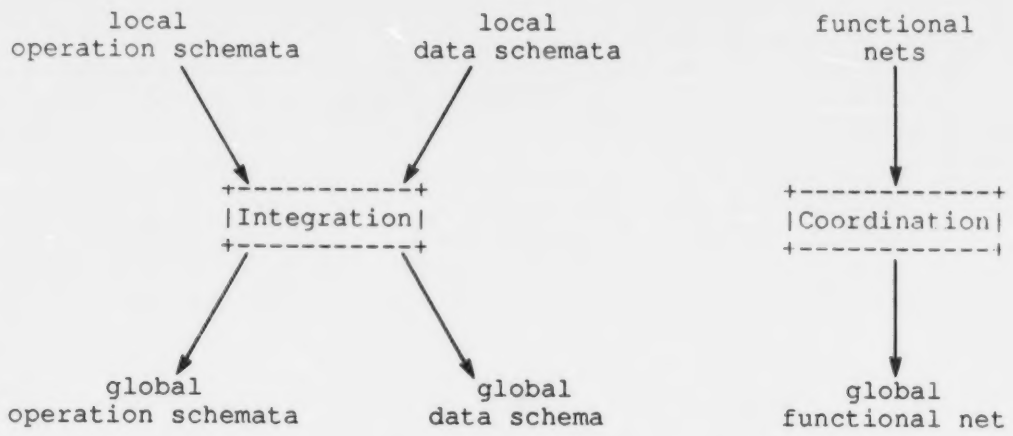


Figure 44: View Integration Overall Architecture

```
[integration]
1. integrate the local data schemata
   1.1 choose a local data schema as the draft data schema
   repeat 1.2 choose a local data schema
          1.3 find concepts in common with the draft data schema
          1.4 resolve conflicts
          1.5 merge schemata
   until  1.6 all local data schemata have been processed
2. restructure the operation schemata

[coordination]
3. coordinate the local functional nets
   3.1 choose a local net as the draft functional net
   repeat 3.2 choose a local net
          3.3 find external conditions in common with the draft
              functional net
          3.4 coordinate the nets
   until  3.5 all local functional nets have been processed
```

Figure 45: View Integration Activities

representations. Moreover, local views can be developed, in general, by different designers at different times. As a consequence, data schema integration is a critical task (37).

The basic activity of the Integration process is the *conflict analysis*, i.e., the resolution of any situation such that a transformation of the integrating schemata is needed to produce the integrated schema.

Conflicts may involve the name of the concepts (*homonym* and *synonym* problems) and their structure (concepts with the same name that have different types or different properties in the two schemata).

In ref. 37, a set of possible solutions is suggested for each case. For instance, with regard to the naming conflicts, they can be discovered by analysing the properties of the concepts and the relationships in which they are involved. If concepts with different names have several properties in common in the two schemata, this can be considered as a synonym indicator, and the conflict can be resolved by introducing a common name or by introducing a subset hierarchy between the concepts. If concepts with the same name have different properties in the two schemata, this can be considered as a homonym indicator, and the conflict can be resolved by introducing different names or by introducing a partition hierarchy.

With regard to the structuring conflicts, the most common situation concerns a concept that is represented as an attribute in one schema and as an entity in another schema. Usually, this conflict is resolved by transforming the attribute into the corresponding entity.

In any case, the transformations to be applied must be discussed with the users involved, in order to get a clearer and more understandable representation.

Once conflicts have been resolved, merging of the two schema can take place by means of a simple superposition of common concepts.

In the example, the resulting global data schema for the FMS organization is shown in Fig. 46.

The local operation schemata must then be restructured according to the transformation applied on the data.

In the activity 3, environment functional nets are coordinated into the global *functional net*. Functional nets are analyzed in the context of the global organization to find synchronous and asynchronous communications (both data and messages) between them.

It must be noted that the resulting global functional net, in which each environment is represented as a macro function, must reproduce the original organization net constructed in the Enterprise Analysis and Modelling phase. The basic difference consists of the complete specification of the "interfaces" between the environments in terms of data and messages which are represented in the global data schema.

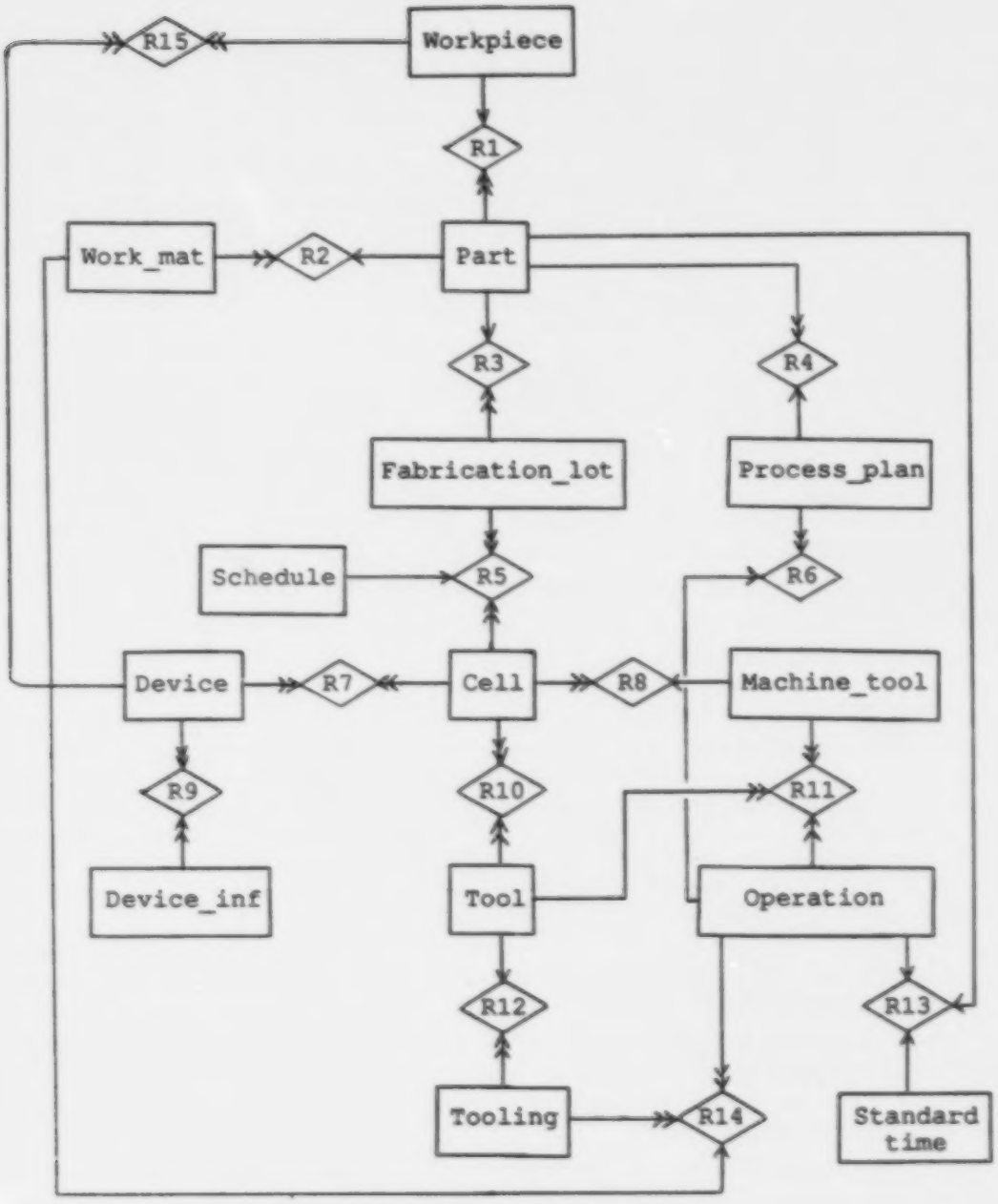


Figure 46: The FMS Global Data Schema

8.6. Validation of Data Schemata in M*

Validation of data schemata is an important feature of any database design methodology. In M*, the completeness and accuracy of data schemata can be achieved by means of:

- The operation schema: They are used to verify the completeness of data schemata in terms of static data requirements, i.e., they are used to verify that all entities, relationships, and attributes have been defined according to the enterprise information objects to be modelled.
- The process nets and functional nets: They are used to verify the good conformity of the data schema in terms of dynamic data requirements, i.e., they are used to ensure that well-formed relationships, right functionalities, sufficient integrity constraints, etc., have been defined according to function operations constraints.

8.7. Conclusion

In this Section, we have presented the M* approach for conceptual design of enterprise database. Materials of this Section are largely based on concepts used in the Italian DATAID methodology. These concepts are among the most advanced tools currently available for conceptual database design (EER model, operation model, and process model). The important features of both DATAID and M* come from the fact that they allow a precise and complete analysis of both static and dynamic data requirements and that they provide tools for validation of the data schemata obtained. All these tools are supported by simple but efficient graphical formalisms.

Once the global database schema has been obtained, the database committee must consider the selection of a commercial database management system (DBMS) for the implementation of the database application (if no system is already being used in the company). This phase is not considered in the M* methodology since it is assumed that the software package DBS/R will be used for database management. However, since Phase 1 and Phase 2 of the methodology are independent of any commercial DBMS and since DBMS software selection is an important task in a database development project, the interested reader can consult ref. 38 to obtain detailed information regarding the process of selecting a commercial DBMS. Note that DBMS selection must be done only after the Enterprise Modelling and Analysis Phase and after the Database Conceptual Design phase because then the database committee will have an exact idea of the DBMS requirements.

The next Phase of the methodology is concerned with database implementation in terms of a physical DBMS implementation environment, DBS/R, which is a relational-based database environment.

9. Phase 3: IMPLEMENTATION DESIGN

9.1. Introduction

The Implementation Design phase is a performance-driven process that maps the conceptual description of the enterprise into a logical-physical description that can be directly processed by the target DBMS chosen for the implementation.

In the Logical Design task, the global data schema is transformed into a *relational schema* in which data structures are expressed according to the logical data model of the DBS/R relational system, i.e., the DBMS chosen for the implementation. In the same way, global operation schemata are converted into *program fragments*.

After that, the Physical Design task constructs a *physical schema*, which specifies the access paths. Criteria are given for choosing an "efficient" logical-physical representation among several alternatives that can be derived from the conceptual schema.

The overall architecture of the Implementation Design phase is shown in Fig. 47.

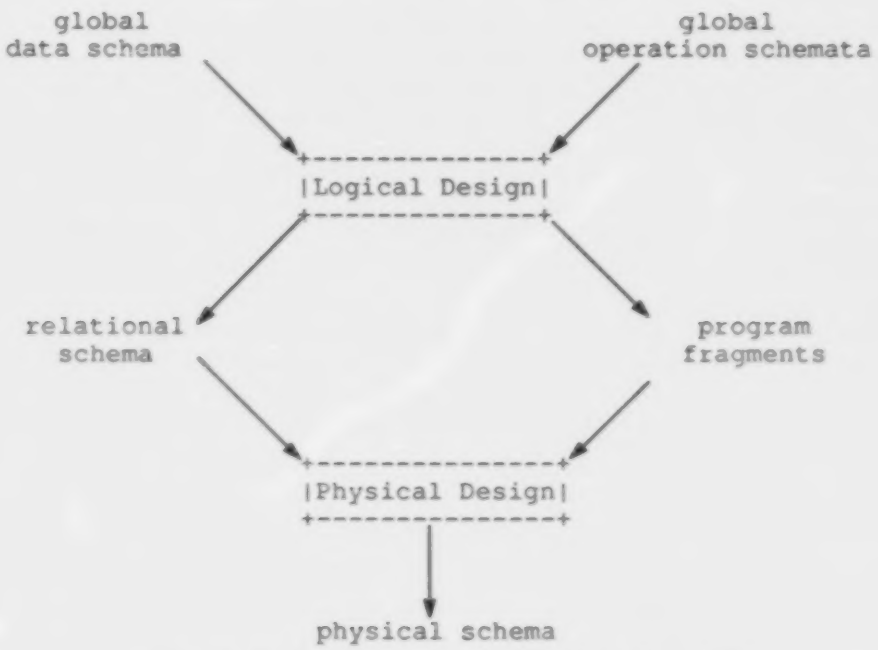


Figure 47: Implementation Design Phase Overall Architecture

9.2. Task 3.1: Logical Design

The main objective of the Logical Design task is to transform the global data schema into a logical data schema in which data structures are expressed according to the relational data model used in the target DBMS, namely DBS/R.

The transformation process is based on two fundamental steps (26):

a. Simplification of the global data schema: Data structures that cannot be directly converted into the relational data model (abstraction hierarchies) are transformed into entities and binary relationships (relationships between two entities).

b. Refinement of the simplified data schema: A set of local restructurings on the objects of the data schema is applied in order to select, among different alternatives, the solution which optimizes the execution of the most important operations.

Since the transformation process is performance-oriented, a first activity collects and analyzes a set of *access parameters* on data and operations. Such information is used to evaluate the number of logical accesses of the operations to the entities and the amount of data transferred in I/O actions between the database and the application program. In the case of DBS/R, mass storage is resident in the virtual memory of the computer (27).

Subsequent activities are the generation of the relational schema and the preparation of program fragments which describe manipulations on data. The architecture of the Logical Design task is shown in Fig. 48.

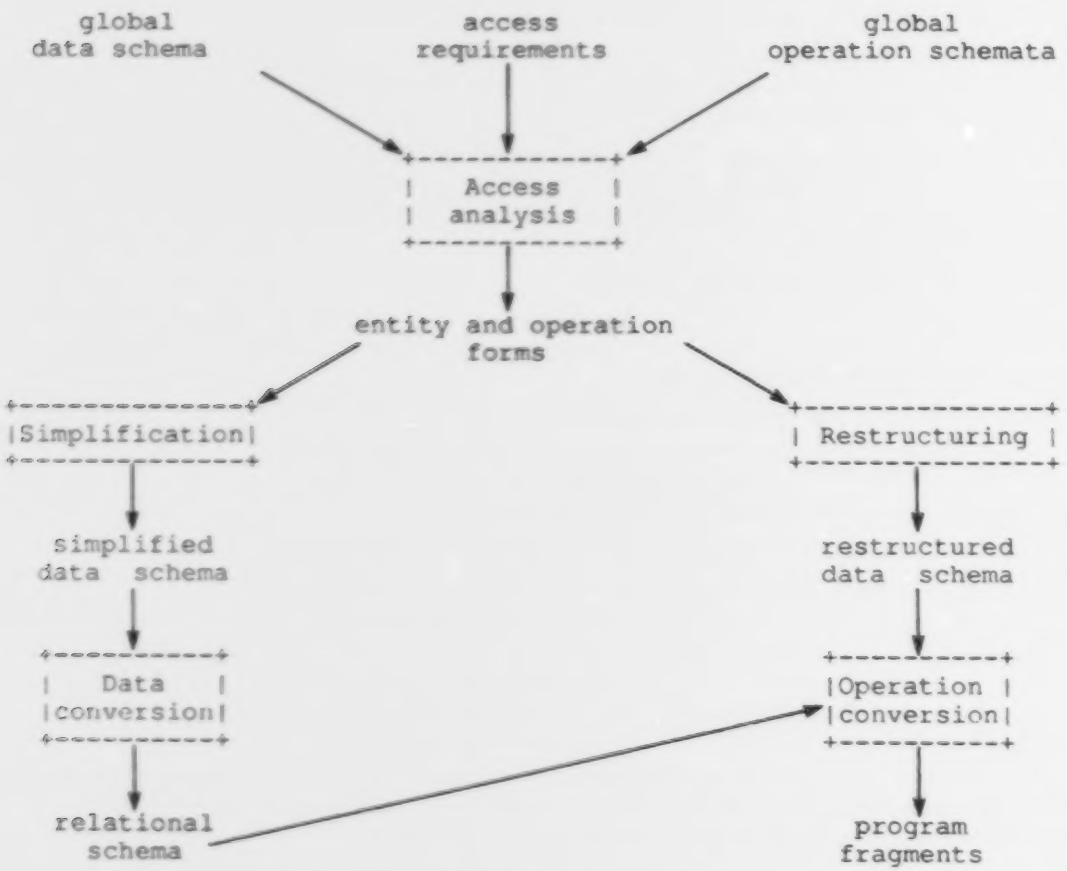


Figure 48: Logical Design Overall Architecture

9.2.1. Task 3.1: Input Description

The input to the Logical Design task consists of the global data schema, the global operation schemata, and the access requirements.

Data and operation schemata are described by means of the EER model. Access requirements are collected from the users involved and then integrated with the data and operation schemata.

Access requirements consist of three types of data:

1. *Access path description*: The access path describes the navigation of the operation within the data schema. The designer has to choose the *access type* for each entity needed by the operation. Relationships are usually accessed by navigation (nv). We can distinguish the following access types for the entities:
   a. *Direct* (ky:<key_name>): The key <key_name> is used to access one occurrence of the entity.
   b. *Sequential* (sq): All the occurrences of the entity must be considered.
   c. *Selection* (sl:<pred>): The subset of the occurrences of the entity which satisfies a given predicate <pred> must be selected.
   d. *Navigational* (nv:<rel_name>): The entity is accessed by means of the relationship <rel_name>.
2. *Usage type description*: A usage type specifies the way the entity or the relationship is used.

   The usage types for entities are the following:
   a. *Read* (rd:<l>): The occurrences are retrieved for query or output purposes. The list l of the attributes to be considered must be supplied.
   b. *Update* (<t>:<l>): The occurrences of the entity are either inserted (<t>=in), or deleted(<t>=dl), or replaced(<t>=rp). The argument <l> of the update operator is the list of the attributes to be inserted or replaced (it is not meaningful for the delete operator). A replace operator cannot change an entity identifier.

   The usage types for relationships are the following:
   a. *Read* (rd:<l>): The occurrences of the relationships are retrieved for query or output purposes. This type of usage is meaningful only for relationships which have attributes. The list l of the attributes to be considered must be supplied.
   b. *Update* (<t>:<l>): The occurrences of the relationship are either inserted, i.e., the entity occurrences are connected (<t>=cn), or deleted, i.e., the entity occurrences are disconnected (<t>=ds). The argument <l> is the list of the attributes to be inserted.
3. *Cardinality information*, which refers to the following quantities:
   a. Frequency of the operations, i.e., the expected number of executions in the unit of time.
   b. Cardinality of entities and relationships, i.e., their average number of occurrences.
   c. Cardinality of the access set (or *access number*), which specifies for each access the average number of occurrences involved in the operation. For direct (ky) access type this number is 1, while for sequential (sq) access type the access number is the entity cardinality. For selection (sl) access type, the selected set cardinality must be evaluated, taking into account the probability distribution of the values of the attributes involved in the selection predicate.

For each access, a complete specification is given by listing the above mentioned requirements. For instance, the requirement [ky:device#; rd:status, location; 1] describes a direct access to the entity 'Device' which uses the identifier 'device#' and reads the attributes 'status' and 'Location'. The access number is 1.

Due to their particular type, quantitative requirements do not usually present semantic interpretation problems.

9.2.2. Task 3.1: Output Description

The output of this task is composed of the relational schema and the program fragments descriptions.

The relational schema is expressed by means of the relational data definition language used by the target DBMS DBS/R. Program fragments are described by means of the relational data manipulation language of the DBS/R system. Both languages have been presented in Section 5.4.

9.2.3. Task 3.1: Description of Activities

The procedural description of the Logical Design activities is shown in Fig. 49.

```
[access analysis]
1. collect access data
2. construct entity forms and operations forms

[simplification]
3. convert hierarchies

[restructuring]
4. restructure
   4.1 merge entities
   4.2 duplicate attributes
   4.3 split entities

[data conversion]
5. convert the global data schema
   5.1 translate entities
   5.2 translate relationships

[operation conversion]
6. translate operation schemata
```

Figure 49: Logical Design Activities

In the access analysis activity, access data are collected. For each entity (and relationship which has attributes) an *entity form* is constructed which contains:

1. The entity description with the attribute names and the corresponding domains, and the entity cardinality;
2. An *access matrix*, which describes, for each operation which makes use of the entity, the access type, the usage type, and the access workload, i.e., the average number of accesses in the unit time (it is obtained as the product of the access number and the operation frequency);
3. A *usage matrix*, which is a logical matrix that describes the attributes that are used by the the reading operations.

Hierarchical entities are represented in a similar way by introducing an access matrix for the superentity and each subentity.

In Fig. 50, a typical entity form is displayed.

For each operation, an *operation form* is constructed which contains:

1. The operation name and frequency (referred to a common unit of time);
2. The *procedural operation schema*: It is the operation schema in which the description of the access on entities and relationships has been specified near the corresponding data object. Navigation in the schema is simply expressed by means of arrows;
3. A *navigation matrix*, which describes, for each entity (or relationship possessing attributes) that is accessed by the operation, the access type, the usage type, and the access workload. This structure is "orthogonal" to the access matrix for the entities.

In Fig. 51, a typical operation form is displayed.

Entity name: Tool (E13) Cardinality: 100

Attributes:	Domains:
a1: tool_code	d1: A(8)
a2: status	d2: int(1)
a3: tool_life	d3: int(2)
a4: location	d4: A(10)
Identifiers:	Domains:
i1: tool#	id1: int(2)

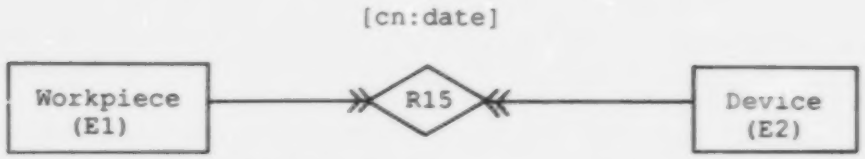
access matrix	T1	T2	T3	T4	T5	T6	T7
access type	s	s	s	s	s	s	k
usage type	rd	rp	rd	rd	rp	rd	rp
workload	500	3200	200	20	250	2000	50

usage matrix	T1	T2	T3	T4	T5	T6	T7
i1							*
a1	*		*	*	*		
a2		*		*		*	
a3		*			*		*
a4	*			*	*	*	

Figure 50: The Tool (E13) Entity Form

Operation_name: Mount (T9) Frequency: 300/day

Procedural operation schema:



Navig. matrix	E1	R15	E2
access type	ky	nv	ky
usage type	rp	cn	rp
workload	300	300	300

Figure 51: The Mount (T9) Operation Form

The Simplification activity converts the abstraction hierarchies by introducing entities and binary relationships (Fig. 52). The attribute t in the simplification of partition hierarchy is a *discriminating attribute* which allows the distinction between occurrences of the type P1 and occurrences of the type P2 (t can be an attribute of the superentity E).

As indicated in Fig. 52, a partition hierarchy can be simplified in three different ways:

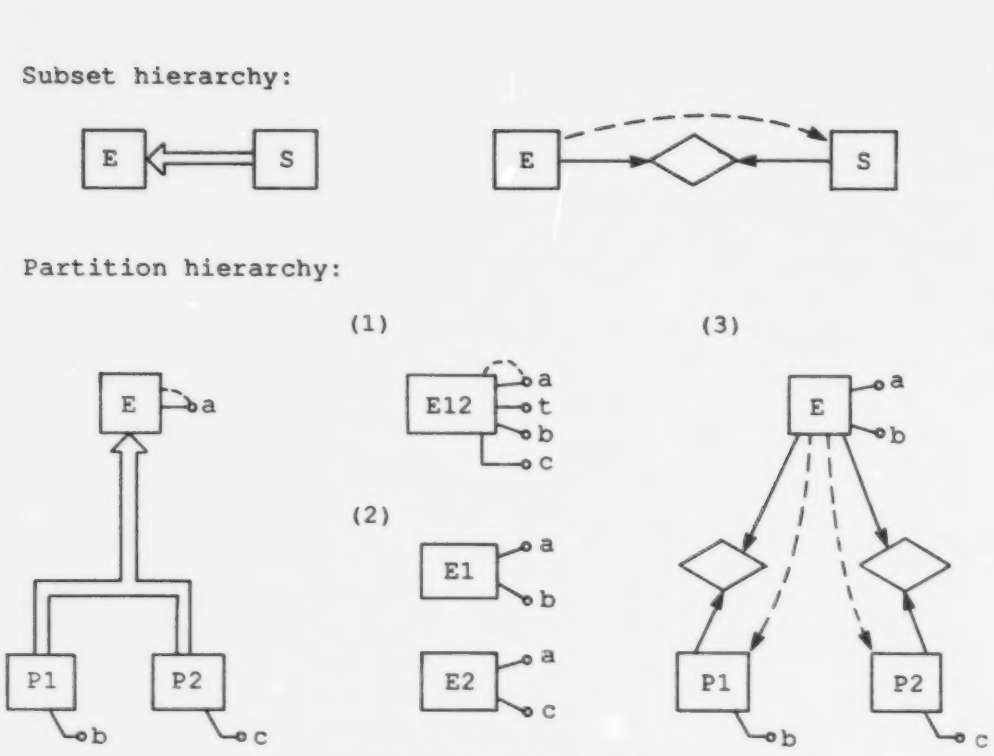


Figure 52: Simplification of the Abstraction Hierarchy

- Solution (1) must be chosen if operations usually access *both* subentities P1 and P2 and the superentity E.
- Solution (2) must be chosen if operations usually access just one of the subentities and the superentity.
- Solution (3) must be chosen if operations usually access only one of the subentities or the superentity.

The choice can be made by simply inspecting the usage matrix of the hierarchized entity, taking into account the workload of each operation.

The restructuring activity performs the following simple refinement steps on the simplified schema (39):

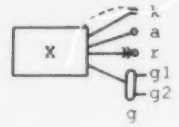
1. Merging: The merging step inspects pairs of entities connected by a single 1:1 relationship. Entities are merged if the operations use, in general, the relationships and *both* the entities. Merging of entities connected by 1:n or n:m relationships is not considered because it involves complex restructurings;
2. Duplicating attributes: In this step, an attribute of an entity is replicated into another entity (connected to the first one) if this redundancy reduces the number of logical accesses of the operations;
3. Splitting: The splitting step is carried out by partitioning attributes into clusters so that disjoint clusters are used by different sets of operations. Each cluster forms a subentity. Subentities are connected by means of 1:1 relationships. The splitting of an entity reduces the amount of data transferred between secondary and central memory in the case of conventional database systems.

The Data Conversion activity translates the entities and the relationships of the restructured data schema into the relational model. The basic translation rules are summarized in Fig. 53.

In the example, the resulting relational schema corresponding to the conceptual schema of Fig. 46 is shown in Fig. 54.

EER structures | Relational conversion

entities and attributes:

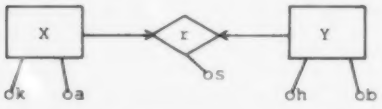


X(k,a,g[g1,g2],{r})

(g[..] indicates a group attribute, and {..} a repeating attribute)

relationships:

a) 1:1 type

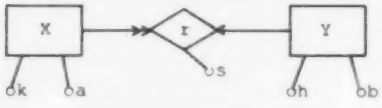


(1) X(k,a) Y'(h,k,b,s)

(2) X'(k,h,a,s) Y(h,b)

(1 and 2 are alternative solutions)

b) 1:n type

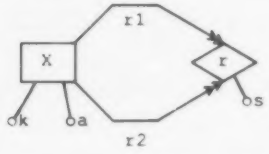


X(k,a) Y'(h,k,b,s)

c) m:n type

X(k,a) Y(h,b) r(k,h,s)

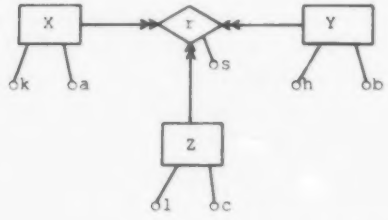
d) ring type



X(k,a) c(kr1,kr2,s)

(r1 and r2 are the roles the entity X has in r: kr1 and kr2 are the corresponding key values)

e) multiple type



X(k,a) Y(h,b) Z(l,c)
r(k,h,l,s)

Figure 53: EER - Relational Translation Rules

- Part (*part_id*, material_code, process_plan#, alternate_process_plan#, NC_program#, alternate_NC_program#, inspection_program#, GT_number)
- Workpiece (*workpiece#*, part_id, status, location, lot#)
- Work_material (*material_code*, material_name, Brinell_hardness_min, Brinell_hardness_max, pretreatment)
- Tooling (*tool_code*, tool_material, tool_material_type, max_speed, min_speed, max_feed, min_feed, max_depth_of_cut, min_depth_of_cut, width_of_cut, length_of_cut, average_life, tool_geometry, tool_diameter)
- Tool (*tool#*, tool_code, status, location, tool_life)
- Operation (*operation_code*, operation_name, description)
- Tool/operation (*tool_code*, *operation_code*, *work_material_code*, *cut_type*, depth_of_cut, speed, feed)
- Machine-tool (*machine#*, cell_id, machine_name, min_speed, max_speed, min_feed, max_feed, min_tool_diameter, max_tool_diameter, status)
- Machine-tool/operation (*machine#*, *operation_code*)
- Fabrication_lot (*lot#*, part_id, quantity, priority, start_date, start_time, cumulated_time, due_date, status)
- Device (*device#*, cell_id, device_code, status, location)
- Device_inf (*device_code*, device_name, mounting_instructions)
- Device_alloc (*device#*, *workpiece#*, date)
- Process_plan (*operation#*, operation_description, department#, cell_id, machine#, tool_code, setup_time, labor_time, number_of_persons, rate_code)
- Standard_time (*operation_code*, *part_id*, setup_time, labor_time, queue_time)
- Schedule (*cell_id*, lot#, start_time)
- Cell (*cell_id*, description, foreman)

Note: Words in italics indicate the keys of relations

Figure 54: The FMS Relational Data Schema

The operation conversion activity translates the operation schemata into the relational data manipulation language of the DBS/R system.

Figure 55 shows the basic translation rules.

Figure 56 shows the program fragment corresponding to the Mount operation.

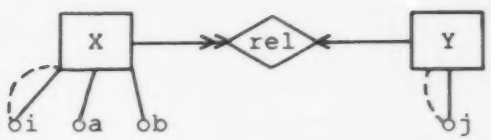
9.3. Task 3.2: Physical Design

The objective of the Physical Design task is to construct, for each relation, a set of secondary indexes which reduces the access time for the database workload.

The index selection problem is strictly dependent on the specific DBMS used to implement the DB application. The DBS/R system allows basically three types of access:

1. A *primary access path* which is based on a hashing function;
2. A *sequential access path* which is based on a bidirectional chain that links together all the tuples of a relation;

EER Data Schema:



Relational Data Schema:

X(i,a,b) Y(j,i)

variable declarations: range of x is X
range of y is Y

EER Access Description:	Relational Access Description:
[ky:i;rd:a,b]	retrieve (x.a, x.b) where i='...'
[ky:i;in:i,a,...]	append to X(i='...', a='...', ...)
[ky:i;dl]	delete x where i='...'
[ky:i;rp:a='...']	replace x(a='...') where i='...'
[sq;rd:a,b]	retrieve (x.a,x.b)
[sq;rp:a='...']	replace x(a='...')
[sl:a='...';rd:i,b]	retrieve (x.i, x.b) where a='...'
[sl:a='...';dl]	delete x where a='...'
[sl:a='...';rp:b='...']	replace x(b='...') where a='...'
[nv:rel;rd:a,b]	retrieve (x.a, x.b) where x.i=y.i

Figure 55: Operation Translation Rules

3. A *secondary index access path* which is based on indexes structured as B+ trees. Each index contains, in the leaves, all the key values, each followed by the set of tuple identifiers (TIDs) where the value appears. A secondary index can be used both for direct access and sequential access according to the order of the keys (in fact, leaf pages are chained together so that the index can be scanned without traversing the tree).

The memory organization for a generic relation is shown in Fig. 57.

The Physical Design is derived from the methodology exposed in ref. 40. It is based on the following assumptions:

a. At most one index is used to access tuples during an operation;

b. Joins are performed according to the nested loops method;

c. An index cannot be used to access tuples if it is currently being updated.

The first activity of the task specifies the database and workload characteristics. Then, the costs of the execution of each operation for each possible access path are evaluated. The index selection is based on cost

EER Operation Schema:

Relational Data Schema:

```
Workpiece (workpiece#, part_id, status, location, lot#)
Device (device#, cell_id, device_code, status, location)
Device_alloc (device#, workpiece#, date)
```

```
<range of> w <is> Workpiece
<range of> d <is> Device
   <replace> w (status='...') <where> w.workpiece#='w1'
   <replace> d (status='...') <where> d.device#='d1'
   <append to> Device_alloc (device#='d1', workpiece#='w1',
       date='...')
```

Figure 56: The Program Fragment for the Mount Operation

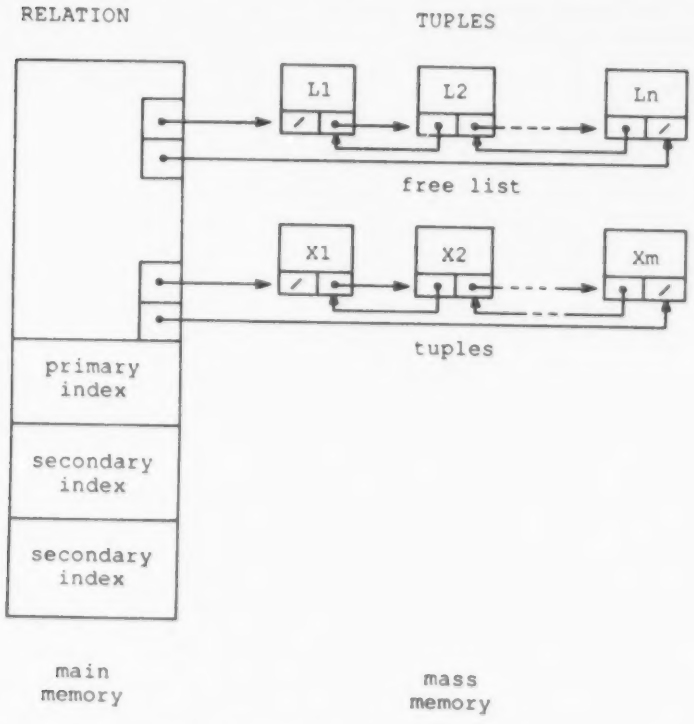


Figure 57: Memory Organization of a Relation

comparisons, in order to determine those indexes that are obviously useless or provide minor benefits when others are present.

The architecture of the Physical Design task is shown in Fig. 58.

9.3.1. Task 3.2: Input Description

The relational schema and the relational operations are specified by means of the relational data definition and data manipulation languages.

9.3.2. Task 3.2: Output Description

The output of the task consists of the list of selected indexes for each relation.

9.3.3. Task 3.2: Description of Activities

The procedural description of the Physical Design activities is shown in Fig. 59.

In the first activity, the characteristics of the database application must be specified.

First of all, the operations are decomposed, i.e., starting from assumption b, queries that involve join operators are separated into single relation subqueries. To explain the join separability concept, let us consider the example of Fig. 60.

In this way, operations can be grouped to characterize a workload for each relation independently.

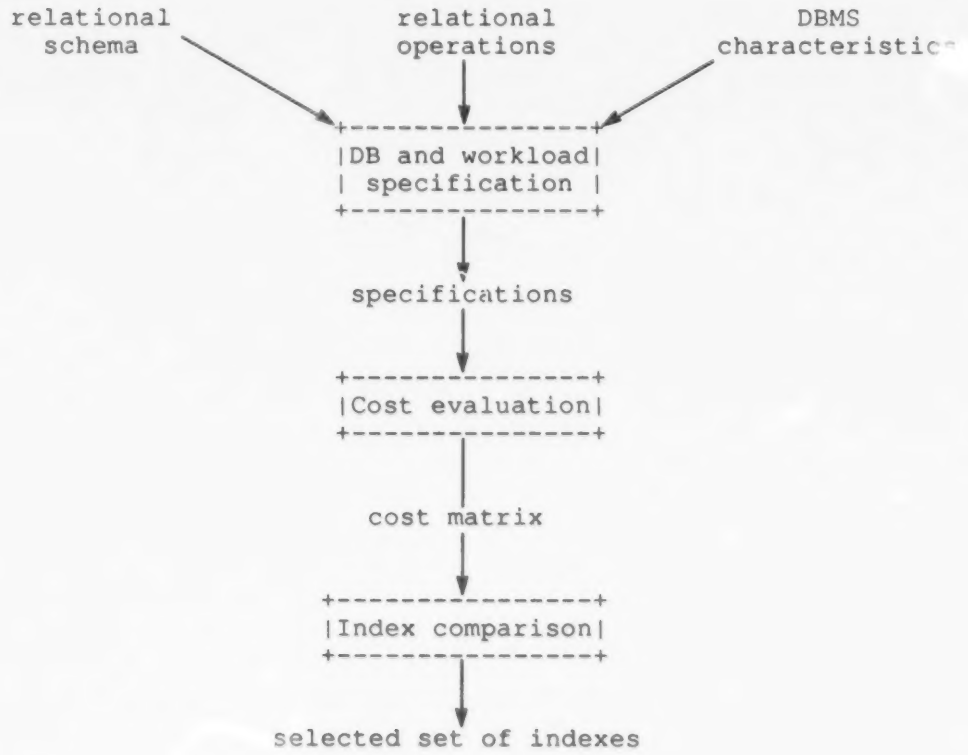


Figure 58: Physical Design Task Overall Architecture

```
[DB and workload specification]
1. specify workload characteristics
   1.1 decompose complex requests
   1.2 evaluate predicate selectivities
2. specify relation characteristics
   2.1 specify storage allocation parameters
   2.2 determine candidate indexes

for each relation

 [cost evaluation]
 3. construct the cost matrix
    for each operation
     3.1 evaluate search costs
    end for
    3.2 evaluate global costs

 [index comparison]
 4. analyze the cost matrix
 5. generate the "best" index set

end for
```

Figure 59: Physical Design Activities

```
Data schema: X(k,a,b) Y(h,k,c,d)

Database variables: range of x is X
                    range of y is Y

Original query:

Q: retrieve (x.a, y.c) where x.b='...' and x.k=y.k and y.d='...'

freq(Q)=f

Subqueries:

Q1: retrieve (x.k,x.a) into T where x.b='...'

freq(Q1)=f

For each tuple [t] of the intermediate result T, the relation
Y must be scanned. Therefore, the second subquery is:

Q2: retrieve (y.b) where y.d='...' and y.k=[t.k]

freq(Q2)=|T|*f

where |T| is the cardinality of T
```

Figure 60: Decomposition of Relational Operations

The selectivity of each predicate that appears in the where clause of an operation is the cardinality of the subset of tuples which satisfy the predicate. This selectivity can be evaluated if the expected probability distribution of the attribute values is taken into account.

In the index selection process, for each relation only the operations that contain a predicate selection on a non-key attribute are considered. In fact, the primary access path on the key and the sequential access path are always present.

A non-key attribute referred to by a predicate selection is called a *candidate index*, because it can be used to speed up the search process if an index is actually associated with it.

The *cost matrix* C(i, j), for a given relation R, is a matrix the rows of which are the operations T(i) (i = 1, ..., n) on the relation, and the columns of which are the candidate indexes S(j) (j = 1, ..., m). An element C(i, j) is the estimated number of physical accesses required to execute the operation i using the index j. C(i, j) can be extimated by means of the following approximated formulas:

a) C(i, j) = |R|
if the index j is not useful

b) C(i, j) = <predicate selectivity>
if the index j is useful

An index j is useful for an operation if the corresponding attribute appears in a selection predicate. Relations a) and b) represent the approximate number of physical accesses to the mass memory, taking into account that, with the memory allocation process of the DBS/R system, tuples are scattered throughout the overall address space. In b) the cost for the index scanning has not been considered because high levels of the indexes are always in central memory. It must be noted that if an operation updates the relation, the secondary indexes must also be updated. In general, the update cost (i.e., the cost of updating the tuples and the indexes) cannot be ignored as shown in ref. 41. Nevertheless, due to the manual nature of the M* methodology, only the access costs of the operations will be taken into account.

To illustrate the above concepts, let us consider the relation Tool. The workload of this relation is shown in Fig. 61.

The corresponding cost matrix is shown in Fig. 62. The row GT(j) represents the global cost (in the unit time) of execution of the object operation set including the index j and the primary index only. This cost is simply given by:

$$GT(j) = \sum_{i=1}^{n} C(i, j) * freq(i)$$

The final activity of this task is the index comparison activity. Following the heuristic procedure explained in ref. 37, the indexes are ordered according to increasing values of the global cost. The comparison can be carried on starting from an *access vector* AV which is initialized with the primary index for each operation. Each one of the candidate indexes is then added to AV in the order previously defined. An index is retained if the global execution cost of all operations with the new AV is lower than the preceding one, otherwise it is eliminated from further considerations. The table in Fig. 63 shows the construction of the AV vector for the relation Tool, starting from the initial value:

AV(j) = {k, k, k, k, k, k} (k = tool# is the primary index)

For each step, a secondary index is added (in the order <x3, x1, x2> of increasing global cost). In the corresponding section of the table, the first row is the access vector, the second row is the corresponding cost vector CV.

As illustrated in the table of Fig. 63, when an index j is added to the access vector, it replaces one or more existing indexes as the best access path for some operations. It may happen that by introducing a new index

Relation: Tool (tool#, tool_code, status, location, tool_life)

Database variable: range of t is Tool

Cardinality: 100

Candidate indexes:{x1=tool_code, x2=status, x3=location}

Workload:

T1 (freq=100):	selectivity	T2 (freq=2):	selectivity
retrieve (....)		replace (....)	
where x1='...'	5	where x2='...'	40
and x3='...'	15		
T3 (freq=80):		T4 (freq=50):	
retrieve (....)		retrieve (....)	
where x1='...'	2	where x1='...'	20
		and x2='...'	10
		and x3='...'	30
T5 (freq=100):		T6 (freq=200):	
replace (....)		retrieve (....)	
where x1='...'	5	where x2='...'	10
and x3='...'	10	and x3='...'	15

Figure 61: Workload Specification for the Relation Tool

	x1	x2	x3	freq
T1	5	100	15	100
T2	100	40	100	2
T3	2	100	100	80
T4	20	10	30	50
T5	5	100	20	100
T6	100	10	15	200
GT	22360	30580	16200	

Figure 62: Cost Matrix for the Relation Tool

some of the existing indexes in the access vector disappear. In the case study, this happens for the index x3 (which is the less expensive index, on the average!). Therefore, the best index set for the relation Tool is:

{k, x1, x2} = {tool#, tool_code, status}

As shown in the table of Fig.63, the global access cost corresponding to the final solution (4240) is the 8% of the initial cost (53200).

Operation:		T1	T2	T3	T4	T5	T6	global
frequency:		100	2	80	50	100	200	cost
	AV:	k	k	k	k	k	k	
	CV:	100	100	100	100	100	100	53200
x3)	AV:	x3	k	k	x3	x3	x3	
	CV:	15	100	100	30	20	15	16200
x1)	AV:	x1	k	x1	x1	x1	x3	
	CV:	5	100	2	20	5	15	5360
x2)	AV:	x1	x2	x1	x1	x1	x2	
	CV:	5	40	2	20	5	10	4240

Figure 63: Generation of the Access Vector

It must be noted that the heuristic which is the basis of the comparison method is motivated by the consideration that most of the indexes with low global execution costs would be included in the optimal solution. But a different order of insertion might yield a more economical solution. It is worth verifying the validity of the solution by using different orderings of insertion.

Moreover, constraints related to mass memory have not been considered. The optimization problem can be managed separately, i.e., on a relation basis, if each relation has a given amount of secondary memory assigned to it, otherwise a global optimization problem must be considered (see ref. 42 for the extension of the comparison method to the global problem).

9.4. Conclusion

This closes this technical Section on logical and physical database implementation with the DBS/R database development system. This also closes the technical discussion on the M* methodology.

At this stage of the general methodological framework for Production and CIM database developement outlined in Section 3.3 (15), we are at the physical design and implementation phase. Thanks to the M* methodology, everything might have now been done concerning the planning and preparation of the database. Therefore, the next stage should be application program development under Database Administrator's supervision, and then the final step would be database operational use and maintenance. These steps are not parts of the M* methodology.

10. CONCLUSION

In this report, we have presented features of the M* methodology for the analysis of information systems of production environments and for the design of the underlying database with a focus on the need for integration of enterprise modelling and analysis with database conceptual design methods. The methodology is still an experimental product now used at the National Research Council of Canada for real-world production environments.

M* has been presented as a methodological tool within a larger methodological framework for implementing Production and CIM databases. However, we know that database design and development is only one step of a larger methodological framework for software project development. Therefore, M* must be considered as one more tool available in software engineering.

Advantages of M* include:

- It is a pragmatic, step by step methodology
- It is complete, i.e., it covers all database design phases
- It is based on good modelling tools (functional models, EER model, Petri nets)
- It deals with static and dynamic properties of data
- It involves a detailed system analysis and modelling phase

However, M* has some disadvantages which are:

- It is a complex manual methodology involving numerous modelling techniques
- It is time-consuming since it is manual and involves a lot of clerical work
- It is prone to human subjectivity for the definition of entities and links among these entities, and during the view integration process

Also, cost analysis is an important factor for the evaluation of a given system or the evaluation of the performance of its functions. Since this has not been considered in our experimentations (we assume that it has been done by managers), it could be of interest to try to augment the set of tools for enterprise modelling and analysis with some cost analysis methods.

The next step is now to undertake extensive experimentation with this methodology. Then, knowledge and expertise acquired on the use of the method can be used to refine it and undertake the implementation of automatic tools to support data analysts in their work during all the analysis and database design phases in order to overcome some disadvantages of the current use of the methodology. More specifically, an expert advisor system could be developed to be used as a tutorial advisor for teaching the principles of the methodology and as an expert analyst for demonstrating the treatment of some experimental case studies. Therefore, theoretical and practical knowledge on the use of the methodology could be obtained from the system. Such a project has been initiated at NRC with national and international participation of universities. In both cases, the expert advisor will deal with details of DBS/R at the implementation design level.

11. ACKNOWLEDGEMENTS

This project was supported in part by the Department of External Affairs Technology Inflow Program.

The authors are grateful to J.W. Brahan, Head of the Information Technology Section of the National Research Council of Canada, who initiated the project and made arrangements for the participation of Prof. A. Di Leva from Italy.

12. REFERENCES

1. M. Groover and E. Zimmers. CAD/CAM Computer-Aided Design and Manufacturing. Prentice-Hall, Englewood Cliffs, NJ. 1984.
2. P. Ranky. Computer-Integrated Manufacturing. Prentice-Hall, Englewood Cliffs, NJ. 1986.
3. F. Vernadat. A Conceptual Schema for a CIM Database. *In* CAD/CAM Integration And Innovation. Society of Manufacturing Engineers, Dearborn, MI. 1985. (*Also in* AUTOFACT VI Conf. Proc., pp. 11.24-11.41).
4. C.J. Date. An Introduction to Database Systems. 4th ed. Addison-Wesley, Reading, MA. 1986.
5. J.D. Ullman. Principles of Database Systems. 2nd ed. Computer Science Press, Rockville, MA. 1982.
6. D.C. Tsichritzis and F.H. Lochovsky. Data Models. Prentice-Hall, Englewood Cliffs, NJ. 1982.
7. T.J. Teorey and J.P. Fry. Design of Database Structures. Prentice-Hall, Englewood Cliffs, NJ. 1982.
8. S.B. Yao (Ed.). Principles of Database Design. Volume I: Logical Organizations. Prentice-Hall, Englewood Cliffs, NJ. 1985.
9. F. Vernadat. Key Technologies for Successful Manufacturing Integration (CIM). CIMTECH Conf. Proc., Boston, MA. 1986. pp. 5-27 to 5-47.
10. C.M. Savage (Ed.). A Program Guide For CIM Implementation. Society of Manufacturing Engineers, Dearborn, MI. 1985.
11. R.R. Bravoco and S.B. Yadav. Requirement Definition Architecture - An Overview. Computers in Industry, 6: 237-251; 1985.
12. H. Maloubier, D. Breuil, G. Doumeingts and J. Gavard. Use of GRAI Method to Analyse and Design Production Management Systems. *In* G. Doumeingts and W.A. Carter (Eds.). Advances In Production Management Systems. North-Holland, Amsterdam. 1984. pp. 127-140.
13. J. Stanley. Modeling your Facility for Modernization: A View of the Modeling Methodologies. CIM-TECH Conf. Proc., Boston, MA. 1986. 6.1-6.7.
14. F. Vernadat. Computer-Integrated Manufacturing: On the Database Aspect. 3rd Canadian CAD/CAM & Robotics Conf. Proc., Toronto, Ont. 1984. pp. 6.1-6.7.
15. F. Vernadat. Appropriate Methodology for Implementing CIM Databases. 5th Canadian CAD/CAM & Robotics Conf. Proc., Toronto, Ont. 1986. Paper MS86-723.
16. E.T. Ross. Structured Analysis for Requirements Definition. IEEE Trans. Software Eng. SE-3(1): 6-15; 1977.
17. C. Gane and T. Sarson. Structured Systems Analysis: Tools and Techniques. Prentice-Hall, Englewood Cliffs, NJ. 1979.
18. T.W. Olle *et al.* Information System Design Methodologies: A Comparative Review. North-Holland, Amsterdam. 1982.
19. S. Navathe and L. Kerschberg. Role of Data Dictionaries in Information Resource Management. Information & Management, 10: 21-46; 1986.
20. R.R. Bravoco and S.B. Yadav. A Methodology to Model the Functional Structure of an Organization. Computers in Industry, 6: 345-361; 1985.
21. V.Y. Lum *et al.* 1978 New Orleans Database Design Workshop Report. IBM Res. Rep. RJ2554 and 5th Int. Conf. on Very Large Data Bases, Rio de Janeiro, Brazil. 1979.
22. A. Albano, V. DeAntonellis and A. DiLeva (Eds.). Computer-Aided Database Design. North-Holland, Amsterdam. 1985.
23. S. Ceri (Ed.). Methodology and Tools for Database Design. North-Holland, Amsterdam. 1983.

24. V. DeAntonellis and A. DiLeva. DATAID-1: A Database Design Methodology. Inf. Syst., 10(2): 181-195; 1985.

25. V. DeAntonellis and A. DiLeva. A Case Study of Database Design Using the DATAID Approach. Inf. Syst., 10(3): 339-359; 1985.

26. V. DeAntonellis and B. Demo. Requirements Collection and Analysis. *In* Ceri S. (Ed.). Methodology And Tools For Database Design. North-Holland, Amsterdam. 1983.

27. DBS/R Reference Manuals. Information Technology Section, National Research Council of Canada, Ottawa, Ontario, Canada K1A 0R8. 1986.

28. B. Demo, A. DiLeva and P. Giolito. Formal Representation of Dynamic Processes. Res. Rep., Dipartimento di Informatica, Universita di Torino, 10125 Torino, Italy. 1986.

29. J.L. Peterson. Petri Net Theory and the Modelling of Systems. Prentice-Hall, Englewood Cliffs, NJ. 1981.

30. P.P. Chen. The Entity Relationship model: Toward a Unified View of Data. ACM Trans. Database Syst., 1(1): 9-36; 1976.

31. J.J. Van Griethuysen *et al.* Concepts and Terminology for the Conceptual Schema and the Information Base. ISO TC97/SC5/WG3 ANSI, New York, NY. 1982.

32. B. Demo, A. DiLeva and P. Giolito. A Predicate Transition Net Based Formalism for Modelling Information System Dynamics. Res. Rep. Dipartimento di Informatica, Universita di Torino, 10125 Torino, Italy. 1986.

33. H.J. Genrich and K. Lautenbach. System Modelling with High Level Petri Nets. Theor. Comput. Sc., 13; 1981.

34. M. Stonebraker *et al.* The Design and Implementation of INGRES. ACM Trans. Database Syst. 1(3): 189-222; 1976.

35. L. Boyer, M. Poirée and E. Salin. Précis d'Organisation Et De Gestion De La Production, Chap. VIII. Les Editions d'Organisation, Paris. 1982.

36. U. Bussolati *et al.* Views Conceptual Design. *In* Ceri S. (Ed.). Methodology And Tools For Database Design. North-Holland, Amsterdam. 1983.

37. C. Batini, M. Lenzerini and M. Moscarini. Views Integration. *In* Ceri S. (Ed.). Methodology And Tools For Database Design. North-Holland, Amsterdam. 1983.

38. F. Vernadat. Selecting the Right Data Base Management System. Can. Datasystems, 17(5): 43-45; 1985.

39. P. Bertaina, A. DiLeva and P. Giolito. Logical Design in CODASYL and Relational Environments. *In* Ceri S. (Ed.). Methodology And Tools For Database Design. North-Holland, Amsterdam. 1983.

40. F. Bonfatti, D. Maio and P. Tiberio. A Separability Based Method for Secondary Index Selection in Physical Database Design. In Ceri S. (Ed.). Methodology And Tools For Database Design. North-Holland, Amsterdam. 1983.

41. M. Schkolnik and P. Tiberio. Estimating the Cost of Updates in a Relational Database. ACM Trans. Database Syst., 10(2); 1985.

42. R. Bonanno, D. Maio, and P. Tiberio. An Approximation Algorithm for Secondary Index Selection in Relational Database Physical Design. The Computer Journal, 28(4); 1985.

43. C. Batini, B. Demo and A. DiLeva. A Methodology for Conceptual Design of Office Data Bases. Inf. Syst., 9(3/4): 251-263; 1984.

44. J. Bubenko and C. Kollhaner. CADIS: Computer-Aided Design of Information Systems. *In* J. Bubenko *et al.* (Eds.). Computer-Aided Information Systems, Analysis And Design. Lund, Sweden. 1972.

13. APPENDIX: LIST OF M* FORMS

In this Section, we present examples of the forms used for the various phases of the methodology.

To facilitate use of the forms and to base the analysis with M* on a structured approach, each form is referenced to the methodological phase and the specific task to which the form is appropriate. The referencing system is in accordance with the numbering used in the report (for example, Phase1. Task 2.2).

We encourage users of the M* methodology to make copies of these forms to support their analysis work.

The list of forms used in M* is as follows:

Enterprise Description:
- 1 - Environment list
- 2 - Environment/functions cross-reference table

Environment Description
- 3 - Functions/users list
- 4 - Forms list
- 5 - Requirements collection form

Conceptual Design
- 6 - Form analysis - Conceptual trees
- 7 - Form analysis - Data glossary
- 8 - Form Analysis - Form schema
- 9 - Requirements analysis - Collection and classification
- 10 - Requirements analysis - Data glossary
- 11 - Requirements analysis - Operation glossary
- 12 - Requirements analysis - Operation schema
- 13 - Requirements analysis - Process net
- 14 - View design - Environment schema
- 15 - View design - Environment functional net
- 16 - View design - Environment operation schema
- 17 - View integration - Global data schema
- 18 - View integration - Global operation schema
- 19 - View integration - Global functional net

Phase1. Task 1.3: Enterprise Description

Date:
Version:
Page of

ENVIRONMENT LIST

Code	Name	Description

Phase 1. Task 1.3: Enterprise Description

Date:
Version:
Page of

ENVIRONMENT/FUNCTIONS CROSS-REFERENCE

Environments:

S :
S :
S :
S :
S :
S :
S :
S :
S :
S :

Functions

F :
F :
F :
F :
F :
F :
F :
F :
F :
F :
F :
F :
F :

Funct. / Envir.	F	F	F	F	F	F	F	F	F	F	F	F	F	F
S														
S														
S														
S														
S														
S														
S														
S														
S														
S														

Phase 1. Task 1.3: Environment Specification

Date:
Version:
Page of

FUNCTIONS/USERS LIST

Environment: S

Functions:

Code	Name	Description	Users code

Users:

Code	Name	Description

Phase 1. Task 1.3: Environment Specification

Date:
Version:
Page of

FORMS LIST

Environment: S

Forms:

Code	Name	Description	Users code

Phase 1. Task 1.3: Environment Specification

Date:
Version:
Page of

REQUIREMENTS COLLECTION

Environment: S

Function: F

User: U

Requirements Description:

Phase 2. Task 2.1: Form Analysis

Date:
Version:
Page of

AREA DESCRIPTION

Environment: S

Function: F

Form: F

User: U

Area Description:

Code	Name	Description

Area Conceptual Trees

Phase 2. Task 2.1: Form Analysis

Date:
Version:
Page of

DATA GLOSSARY

Environment: S

Form: F

Code	Name	Description	Domain	Instance	Synonym	Hierarchy	Reference

Phase 2. Task 2.1: Form Analysis

Date:
Version:
Page of

FORM SCHEMA

Environment: S

Form: F

Phase 2. Task 2.2: Requirements Analysis

Date:
Version:
Page of

COLLECTION AND CLASSIFICATION

Environment: S

Function: F

User: U

Type	Code	Sentence

Phase 2. Task 2.2: Requirements Analysis

Date:
Version:
Page of

DATA GLOSSARY

Environment: S

Function: F

Code	Name	Description	Domain	Instance	Synonym	Subset	Reference

OPERATION GLOSSARY

Environment: S

Function: F

Code	Name	Description	Data Involved	Frequence	Reference

Phase 2. Task 2.2: Requirements Analysis

Date:
Version:
Page of

OPERATION SCHEMA

Environment: S

Function: F

Phase 2. Task 2.2: Requirements Analysis

Date:
Version:
Page of

PROCESS NET

Environment: S

Function: F

Phase 2. Task 2.3: View Design

Date:
Version:
Page of

ENVIRONMENT SCHEMA

Environment: S

Phase 2. Task 2.3: View Design

Date:
Version:
Page of

ENVIRONMENT FUNCTIONAL NET

Environment: S

Phase 2. Task 2.3: View Design

Date:
Version:
Page of

ENVIRONMENT OPERATION SCHEMA

Environment: S

Phase 2. Task 2.4: View Integration

Date:
Version:
Page of

GLOBAL DATA SCHEMA

Phase 2. Task 2.4: View Integration

Date:
Version:
Page of

GLOBAL OPERATION SCHEMA

Phase 2 Task 2.4: View Integration

Date:
Version:
Page of

GLOBAL FUNCTIONAL NET